Prof. Dr. Lutz Wegner **24.10.2003**
Dipl.-Ing. Nabil Benamar

Klausur zur Lehrveranstaltung „UNIX“ im WS 2003/04

Name: ..... Vorname: .....

Matr.Nr.: ..... FB: .....

Bearbeiten Sie alle Fragen! Bei Ankreuzaufgaben kann mehr als eine Antwort richtig sein. Hilfsmittel sind nicht zugelassen.

Aufgabe 1:

In den Anfängen von UNIX war die Anzahl der i-Knoten im Dateisystem auf 2^{16} - 1 beschränkt, weil man den Index mit einem 2-Byte Integer auswählte. Diese Beschränkungen

( ) plagen UNIX auch heute noch.

( ) können heutige Systeme leicht aufheben.

( ) waren nach dem Übergang auf graphische Oberflächen irrelevant.

( ) gelten noch für public domain Systeme, etwa LINUX, nicht aber für die großen kommerziellen Systeme wie AIX und Solaris.

Aufgabe 2:

Wie im Kurs gesehen, gab es das `crypt`-Kommando nicht, die `vi`-Version auf dem Linux-System kannte aber die `-x` Option, die eine Schlüsseleingabe verlangt. Damit kann man eine Datei verschlüsseln, etwa durch Aufruf von `vi -x Klartext` und danach schreiben im Kommandomodus mit `:w Codetext`.

( ) Richtig.

( ) Ja und Nein. Man kann damit den Text verschlüsseln, man kann ihn später aber nicht mehr dekodieren!

( ) Nein, das geht nur mit dem `crypt`-Kommando

( ) Ja, aber nur wie der `crypt`-Systemaufruf auf ein einzelnes Wort anwendbar, nicht auf ganze Texte.

Aufgabe 3:

Bei jedem Aufruf eines Programms mit SUID-Bit und Besitzer `root` wird intern das Passwort des Superusers abgefragt.

( ) Richtig. Teil des `exec`-Systemaufrufs zur Überlagerung des Codes.

( ) Richtig. Das SUID-Bit-Programm liest aus `/etc/passwd`.

( ) War früher richtig, jetzt als Sicherheitslücke geschlossen.

( ) Falsch, macht keinen Sinn.

( ) Falsch, könnte man aber zur Stärkung der Sicherheit bei Kommandos wie `at`, `mknod` usw. einführen.

Aufgabe 4:

Hat man auf einem Rechner mehrere Fenster offen, dann geht eine Standardausgabe, die in `/dev/tty` umgelenkt wird, in das aktuelle, aktive Fenster.

( ) richtig.

( ) falsch, geht nur mit echten Terminals.

( ) falsch, geht an alle Fenster.

( ) richtig, aber nur wenn ein Link mit `ln` von `/dev/tty` auf das Fenster gemacht wurde.

( ) falsch, denn `root` ist Besitzer von `/dev/tty`.

Aufgabe 5:

Wer man ist und an welchem Terminal (Fenster) man gerade angemeldet ist, bekommt man z.B. mit `who am I` heraus. Die Tatsache, daß man das eigene Terminal, etwa `/dev/pts/36`, modifiziert hat, erkennt man am Modifikationsdatum der Ausgabe von `ls -l /dev/pts/36`.

( ) Richtig.

( ) Falsch, da will mich jemand auf den Arm nehmen. Weder gibt es `who am I` noch haben Geräte ein Modifikationsdatum.

( ) Falsch, zwar gibt es `who am I` (es müssen nur 2 Argumente sein, z.B. auch `who are you` oder `who mom likes`), aber ein Modifikationsdatum an Gerätedateien gibt es nicht.

( ) Falsch, `who am I` gibt es nicht, aber alle Dateien in `/dev` haben natürlich auch ein Modifikationsdatum.

Aufgabe 6:

Manche Werte von Shellvariablen kann der Anwender kurzfristig ändern, andere sind read-only markiert, aus ganz einsichtigen Gründen. Welche der folgenden Shellvariablen kann ein Benutzer deshalb sicher **nicht** in einer Sitzung kurz einmal ändern?

( ) HOME

( ) UID

( ) PPID

( ) PATH

( ) PS1

Aufgabe 7:

Geben Sie eine kurze Eingabeliste `Test` mit einem Wort je Zeile an, so daß die Ausgabe von

```
cat Test| tr a-z A-Z | sort
```

ungleich ist mit der Ausgabe von

```
cat Test| sort | tr a-z A-Z
```

Test

Aufgabe 8:

Das `which`-Kommando durchsucht den Kommandopfad nach einem eingegebenen Argument, so wie es die Shell für ein eingegebenes Kommando macht. Bei Mißerfolg produziert z.B. RedHat Linux auf `pool-serv1` den folgenden Wortschwall auf `stderr`:

```
[wegner@pool-serv1 wegner]$ which nonsense
/usr/bin/which: no nonsense in (/opt/intel/compiler60/ia32/
bin:/usr/kerberos/bin:/usr/java/j2sdk1.4.0_03/bin:/opt/intel/
compiler60/ia32/bin:/usr/local/bin:/bin:/usr/bin:/usr/X11R6/
bin:/home/wegner/bin)
[wegner@pool-serv1 wegner]$
```

SuSE Linux auf `haensel` gibt sich dagegen wortkarg und liefert

```
wegner@haensel:/home/staff> which nonsense
wegner@haensel:/home/staff>
```

Das ist

( ) ganz schlimm und typisch für die beginnende Uneinheitlichkeit der „UNIXe“.

( ) OK solange beide wenigstens bei Mißerfolg einen Exit-Status ungleich Null liefern.

( ) OK solange bei Mißerfolg in beiden Fällen `stdout` leer ist.

Aufgabe 9:

Bei `cd ..` kann es sich die Shell leichtmachen, weil die Nummer (i-node) des Vorgängerknotens beim Namen, hier `..`, im Katalog steht, wie bei allen anderen Katalognamen auch. Bei manchen Knoten, z.B. der Wurzel (`/`) kommt man aber nicht höher, sondern bleibt in dem gegenwärtigen Verzeichnis. Welcher Wert steht deshalb anstelle des **?** in der folgenden Ausgabe?

```
$ls -ai / | less
        2 .
        ? ..
       12 .autofsck
    58302 .automount
...
```

Ihre Antwort:

Aufgabe 10:

Das folgende Shell-Skript `kuerzung` soll aus einer Eingabezeile immer nur jedes 2. Wort ausgeben, also bei Eingabe `Rente und Weihnachtsgeld werden gekürzt` die Ausgabe `Rente Weihnachtsgeld gekürzt` liefern. Ergänzen Sie die fehlenden Teile, die prüfen, ob es noch Argumente für die Kürzung gibt.

```
read line
set $line
while [ ________________________ ]
do
  echo -n $1" "
  shift 1
  if [ ________________________]
  then shift
  fi
done
```

Aufgabe 11:

Statt sich klassisch mit CTRL-D abzumelden, verwenden immer mehr verweichlichte Menschen das `logout`-Kommando. Immerhin sollten die wissen, daß

( ) `logout` ein Shell-Skript ist.

( ) `logout` ein eingebautes Kommando des BASH ist.

( ) nach dem Abmelden `login` übriggebliebene Waisenkinder adoptiert.

( ) die Söhne und Enkel eines Shell-Prozesses das Abmelden überleben.

Aufgabe 12:

Mit `export gruss` kann man die Variable `gruss` für den „Export“ in Unterprozesse markieren. Mit `export -n gruss` hebt man die Markierung auf.

( ) Wurde im Kurs nicht besprochen, macht aber Sinn.

( ) Nicht möglich, weil ein Variablenexport bis zum `logout` bestehen bleiben muß.

( ) Nicht möglich und nicht nötig, weil `export` sowieso immer eine Einbahnstraße ist.

( ) Gut möglich, müßte aber dann in den `.profile` Dateien stehen und kann erst nach Neuanmeldung wirksam werden.

( ) Nein, eine `export`-Markierung hebt man mit dem `import`-Kommando auf.

Aufgabe 13:

Wie lautet die Ausgabe von `echo -n "$IFS" | wc` ?

( ) 0 0 0

( ) 0 0 1

( ) 0 0 2

( ) 1 0 3

( ) 1 0 4

Aufgabe 14:

```
[wegner@pool-serv1 wegner]$ PWD=/home/wegner/bin
wegner@pool-serv1 bin]$ echo $PWD
/home/wegner/bin
wegner@pool-serv1 bin]$ pwd
/home/wegner
wegner@pool-serv1 bin]$ cd
wegner@pool-serv1 wegner]$ echo $PWD
/home/wegner
```

Aus dem Dialog oben erkennt man

( ) Eine Wertzuweisung an `PWD` bewirkt keinen Wechsel des Arbeitsverzeichnisses.

( ) `cd` setzt die Variable `PWD.`

( ) Die „Ortsangabe“ vor dem Promptzeichen, hier z.B. `bin` oder `wegner`, muß nicht mit dem realen Arbeitsverzeichnis übereinstimmen.

( ) Die „Ortsangabe“ vor dem Promptzeichen, hier z.B. `bin` oder `wegner`, wird von $PWD gesteuert.

Aufgabe 15:

Welche Ausgabe liefert die folgende Kommandozeile sinngemäß.

```
echo date | (read cmd; exec $cmd)
```

( ) das Wort `date`

( ) das Wort `$cmd`

( ) die Datumszeile, also z.B. `Mon Oct 20 14:37:23 CEST 2003`

( ) die Fehlermeldung `Mon: unknown command`

( ) `login:`

Aufgabe 16:

Zombie-Prozesse entsteht recht selten. Da sie unnötig mit ihrem Abbild (image) Hauptspeicherplatz belegen und Rechenzeit verbrauchen, sollte man sie mit `kill -9` entfernen.

( ) Richtig

( ) Falsch, Zombie-Prozesse haben weder ein Abbild noch verbrauchen sie Rechenzeit.

( ) Richtig und falsch. Sie verbrauchen nur Rechenzeit.

( ) Richtig und falsch. Sie belegen nur Hauptspeicher durch ihr Abbild.

( ) Richtig, kann jeder User mit `kill` (auch ohne `-9`) selbst.

Aufgabe 17:

Im Kurs wird von „entlaufenen Prozessen“ gesprochen, die es zu begrenzen gelte. Das betrifft Prozesse,

( ) die zu viele Kinderprozesse erzeugen.

( ) die zu große Dateien erzeugen.

( ) die zu viel Hauptspeicher anfordern.

( ) die mehr als 9 Argumente in der Kommandozeile haben.

( ) die nicht „`nice`“ sind und Rechenzeit klauen, d.h. deren Priorität zu hoch gesetzt ist.

Aufgabe	**Punkte**	**Bemerkung**	**Aufgabe**	**Punkte**	**Bemerkung**
1			Übertrag		
2			10		ausfüllen
3			11		
4			12		
5			13		
6			14		
7		ausfüllen	15		
8			16		
9			17		
Zwischen-summe			**Summe**		

Prof. Dr. Lutz Wegner **24.10.2003**
Dipl.-Ing. Nabil Benamar

Klausur zur Lehrveranstaltung „UNIX“ im WS 2003/04

Name: ..... **Musterlösung** **Vorname: .....**

Matr.Nr.: ..... FB: .....

Bearbeiten Sie alle Fragen! Bei Ankreuzaufgaben kann mehr als eine Antwort richtig sein. Hilfsmittel sind nicht zugelassen.

Aufgabe 1:

In den Anfängen von UNIX war die Anzahl der i-Knoten im Dateisystem auf 2^{16} - 1 beschränkt, weil man den Index mit einem 2-Byte Integer auswählte. Diese Beschränkungen

( ) plagen UNIX auch heute noch.

(X) können heutige Systeme leicht aufheben.

( ) waren nach dem Übergang auf graphische Oberflächen irrelevant.

( ) gelten noch für public domain Systeme, etwa LINUX, nicht aber für die großen kommerziellen Systeme wie AIX und Solaris.

Aufgabe 2:

Wie im Kurs gesehen, gab es das `crypt`-Kommando nicht, die `vi`-Version auf dem Linux-System kannte aber die `-x` Option, die eine Schlüsseleingabe verlangt. Damit kann man eine Datei verschlüsseln, etwa durch Aufruf von `vi -x Klartext` und danach schreiben im Kommandomodus mit `:w Codetext`.

(X) Richtig.

( ) Ja und Nein. Man kann damit den Text verschlüsseln, man kann ihn später aber nicht mehr dekodieren!

( ) Nein, das geht nur mit dem `crypt`-Kommando

( ) Ja, aber nur wie der `crypt`-Systemaufruf auf ein einzelnes Wort anwendbar, nicht auf ganze Texte.

Aufgabe 3:

Bei jedem Aufruf eines Programms mit SUID-Bit und Besitzer `root` wird intern das Passwort des Superusers abgefragt.

( ) Richtig. Teil des `exec`-Systemaufrufs zur Überlagerung des Codes.

( ) Richtig. Das SUID-Bit-Programm liest aus `/etc/passwd`.

( ) War früher richtig, jetzt als Sicherheitslücke geschlossen.

(X) Falsch, macht keinen Sinn.

( ) Falsch, könnte man aber zur Stärkung der Sicherheit bei Kommandos wie `at`, `mknod` usw. einführen.

Aufgabe 4:

Hat man auf einem Rechner mehrere Fenster offen, dann geht eine Standardausgabe, die in `/dev/tty` umgelenkt wird, in das aktuelle, aktive Fenster.

(X) richtig.

( ) falsch, geht nur mit echten Terminals.

( ) falsch, geht an alle Fenster.

( ) richtig, aber nur wenn ein Link mit `ln` von `/dev/tty` auf das Fenster gemacht wurde.

( ) falsch, denn `root` ist Besitzer von `/dev/tty`.

Aufgabe 5:

Wer man ist und an welchem Terminal (Fenster) man gerade angemeldet ist, bekommt man z.B. mit `who am I` heraus. Die Tatsache, daß man das eigene Terminal, etwa `/dev/pts/36`, modifiziert hat, erkennt man am Modifikationsdatum der Ausgabe von `ls -l /dev/pts/36`.

(X) Richtig.

( ) Falsch, da will mich jemand auf den Arm nehmen. Weder gibt es `who am I` noch haben Geräte ein Modifikationsdatum.

( ) Falsch, zwar gibt es `who am I` (es müssen nur 2 Argumente sein, z.B. auch `who are you` oder `who mom likes`), aber ein Modifikationsdatum an Gerätedateien gibt es nicht.

( ) Falsch, `who am I` gibt es nicht, aber alle Dateien in `/dev` haben natürlich auch ein Modifikationsdatum.

Aufgabe 6:

Manche Werte von Shellvariablen kann der Anwender kurzfristig ändern, andere sind read-only markiert, aus ganz einsichtigen Gründen. Welche der folgenden Shellvariablen kann ein Benutzer deshalb sicher **nicht** in einer Sitzung kurz einmal ändern?

( ) HOME

(X) UID

(X) PPID

( ) PATH

( ) PS1

Aufgabe 7:

Geben Sie eine kurze Eingabeliste `Test` mit einem Wort je Zeile an, so daß die Ausgabe von

```
cat Test| tr a-z A-Z | sort
```

ungleich ist mit der Ausgabe von

```
cat Test| sort | tr a-z A-Z
```

Test

```
armer
Egon
```

Aufgabe 8:

Das `which`-Kommando durchsucht den Kommandopfad nach einem eingegebenen Argument, so wie es die Shell für ein eingegebenes Kommando macht. Bei Mißerfolg produziert z.B. RedHat Linux auf `pool-serv1` den folgenden Wortschwall auf `stderr`:

```
[wegner@pool-serv1 wegner]$ which nonsense
/usr/bin/which: no nonsense in (/opt/intel/compiler60/ia32/
bin:/usr/kerberos/bin:/usr/java/j2sdk1.4.0_03/bin:/opt/intel/
compiler60/ia32/bin:/usr/local/bin:/bin:/usr/bin:/usr/X11R6/
bin:/home/wegner/bin)
[wegner@pool-serv1 wegner]$
```

SuSE Linux auf `haensel` gibt sich dagegen wortkarg und liefert

```
wegner@haensel:/home/staff> which nonsense
wegner@haensel:/home/staff>
```

Das ist

( ) ganz schlimm und typisch für die beginnende Uneinheitlichkeit der „UNIXe“.

(X) OK solange beide wenigstens bei Mißerfolg einen Exit-Status ungleich Null liefern.

(X) OK solange bei Mißerfolg in beiden Fällen `stdout` leer ist.

Aufgabe 9:

Bei `cd ..` kann es sich die Shell leichtmachen, weil die Nummer (i-node) des Vorgängerknotens beim Namen, hier `..`, im Katalog steht, wie bei allen anderen Katalognamen auch. Bei manchen Knoten, z.B. der Wurzel (`/`) kommt man aber nicht höher, sondern bleibt in dem gegenwärtigen Verzeichnis. Welcher Wert steht deshalb anstelle des **?** in der folgenden Ausgabe?

```
$ls -ai / | less
        2 .
        ? ..
       12 .autofsck
    58302 .automount
...
```

Ihre Antwort: 2

Aufgabe 10:

Das folgende Shell-Skript `kuerzung` soll aus einer Eingabezeile immer nur jedes 2. Wort ausgeben, also bei Eingabe `Rente und Weihnachtsgeld werden gekürzt` die Ausgabe `Rente Weihnachtsgeld gekürzt` liefern. Ergänzen Sie die fehlenden Teile, die prüfen, ob es noch Argumente für die Kürzung gibt.

```
read line
set $line
while [ ______$# -ne 0_____ ]
do
  echo -n $1" "
  shift 1
  if [ _______$# -ne 0 ____]
  then shift
  fi
done
```

auch richtig

`$# -gt 1`

oder

`$1 != ””`

dann aber leerer String, nicht Blank

Aufgabe 11:

Statt sich klassisch mit CTRL-D abzumelden, verwenden immer mehr verweichlichte Menschen das `logout`-Kommando. Immerhin sollten die wissen, daß

( ) `logout` ein Shell-Skript ist.

(X) `logout` ein eingebautes Kommando des BASH ist.

( ) nach dem Abmelden `login` übriggebliebene Waisenkinder adoptiert.

( ) die Söhne und Enkel eines Shell-Prozesses das Abmelden überleben.

Aufgabe 12:

Mit `export gruss` kann man die Variable `gruss` für den „Export“ in Unterprozesse markieren. Mit `export -n gruss` hebt man die Markierung auf.

(X) Wurde im Kurs nicht besprochen, macht aber Sinn.

( ) Nicht möglich, weil ein Variablenexport bis zum `logout` bestehen bleiben muß.

( ) Nicht möglich und nicht nötig, weil `export` sowieso immer eine Einbahnstraße ist.

( ) Gut möglich, müßte aber dann in den `.profile` Dateien stehen und kann erst nach Neuanmeldung wirksam werden.

( ) Nein, eine `export`-Markierung hebt man mit dem `import`-Kommando auf.

Aufgabe 13:

Wie lautet die Ausgabe von `echo -n "$IFS" | wc` ?

( ) 0 0 0

( ) 0 0 1

( ) 0 0 2

(X) 1 0 3

( ) 1 0 4

Aufgabe 14:

```
[wegner@pool-serv1 wegner]$ PWD=/home/wegner/bin
[wegner@pool-serv1 bin]$ echo $PWD
/home/wegner/bin
[wegner@pool-serv1 bin]$ pwd
/home/wegner
[wegner@pool-serv1 bin]$ cd
[wegner@pool-serv1 wegner]$ echo $PWD
/home/wegner
```

Aus dem Dialog oben erkennt man

(X) Eine Wertzuweisung an `PWD` bewirkt keinen Wechsel des Arbeitsverzeichnisses.

(X) `cd` setzt die Variable `PWD.`

(X) Die „Ortsangabe“ vor dem Promptzeichen, hier z.B. `bin` oder `wegner`, muß nicht mit dem realen Arbeitsverzeichnis übereinstimmen.

(X) Die „Ortsangabe“ vor dem Promptzeichen, hier z.B. `bin` oder `wegner`, wird von $PWD gesteuert.

Aufgabe 15:

Welche Ausgabe liefert die folgende Kommandozeile sinngemäß.

```
echo date | (read cmd; exec $cmd)
```

( ) das Wort `date`

( ) das Wort `$cmd`

(X) die Datumszeile, also z.B. `Mon Oct 20 14:37:23 CEST 2003`

( ) die Fehlermeldung `Mon: unknown command`

( ) `login:`

Aufgabe 16:

Zombie-Prozesse entsteht recht selten. Da sie unnötig mit ihrem Abbild (image) Hauptspeicherplatz belegen und Rechenzeit verbrauchen, sollte man sie mit `kill -9` entfernen.

( ) Richtig

(X) Falsch, Zombie-Prozesse haben weder ein Abbild noch verbrauchen sie Rechenzeit.

( ) Richtig und falsch. Sie verbrauchen nur Rechenzeit.

( ) Richtig und falsch. Sie belegen nur Hauptspeicher durch ihr Abbild.

( ) Richtig, kann jeder User mit `kill` (auch ohne `-9`) selbst.

Aufgabe 17:

Im Kurs wird von „entlaufenen Prozessen“ gesprochen, die es zu begrenzen gelte. Das betrifft Prozesse,

(X) die zu viele Kinderprozesse erzeugen.

(X) die zu große Dateien erzeugen.

(X) die zu viel Hauptspeicher anfordern.

( ) die mehr als 9 Argumente in der Kommandozeile haben.

( ) die nicht „`nice`“ sind und Rechenzeit klauen, d.h. deren Priorität zu hoch gesetzt ist.

Aufgabe	**Punkte**	**Bemerkung**	**Aufgabe**	**Punkte**	**Bemerkung**
1	2		Übertrag		
2	2		10	4	ausfüllen
3	2		11	2	
4	2		12	2	
5	2		13	2	
6	2		14	2	
7	4	ausfüllen	15	2	
8	2		16	2	
9	2		17	2	
Zwischen-summe			**Summe**	**max 38**	**bestanden ≥ 17**

Dipl.-Ing. Nabil Benamar
Dipl.-Math. Kai Schweinsberg

22. Juli 2004

Einführung in UNIX

Klausur

Name: ..... Vorname: .....

Matr. Nr.: ..... Fachbereich:.....

Bearbeiten Sie alle Fragen!
Bei Ankreuzaufgaben muss mindestens eine Antwort angekreuzt werden. Sind alle Antworten angekreuzt, so wird die Aufgabe nicht gewertet.
Hilfsmittel sind nicht zugelassen!
Falls Sie für die Beantwortung der Aufgaben zusätzlichen Platz benötigen, verwenden Sie die Rückseiten. Separate Blätter sind nicht zugelassen.

Aufgabe	Mögliche Punkte	Erreichte Punkte
1	2,0	
2	4,0	
3	2,0	
4	4,5	
5	2,0	
6	2,5	
7	2,0	
8	2,5	
9	3,0	
10	2,0	
11	2,0	
12	3,0	
13	2,5	
14	3,0	
15	3,0	
16	4,0	
17	4,0	
Summe	**48,0**	

Viel Erfolg!

Aufgabe 1: Was versteht man unter einem „T-Stück“?

- ☐ Das Setzen zweier Hard Links auf dasselbe Verzeichnis.
- ☐ Den Einsatz mehrerer Terminals an einem Rechner (nur in älteren UNIX-Versionen).
- ☐ Das Verwenden des Kommandos `tee` innerhalb einer Pipeline.
- ☐ Die Übergabe der Werte aller Shell-Variablen an eine Untershell durch das Kommando `make_t`.

Aufgabe 2: Welche der folgenden Aussagen zur Passwort-Datei `/etc/passwd` sind richtig?

- ☐ Jeder Benutzer kann nur seinen eigenen Eintrag in der Passwort-Datei auflisten, die Einträge der anderen Benutzer bleiben aus Sicherheitsgründen verborgen.
- ☐ Nur der Super-User kann die Datei lesen.
- ☐ Jeder Benutzer des Systems kann die Datei lesen.
- ☐ Jeder Benutzer kann sein Passwort durch Editieren der Datei ändern (Alternative zum Kommando `passwd`).
- ☐ In der Datei wird jedem Benutzer ein Heimatverzeichnis zugeordnet.
- ☐ Die Passwort-Datei legt fest, welchen Gruppen die einzelnen Benutzer angehören.
- ☐ Aktuelle UNIX-Versionen speichern in der Passwort-Datei aus Sicherheitsgründen keine Passwörter mehr.
- ☐ Durch das Hinzufügen eines Punkts am Ende eines Eintrags kann festgelegt werden, dass das jeweilige Arbeitsverzeichnis in den Suchpfad des Benutzers einbezogen wird.

Aufgabe 3: Welche Aussagen zu Hard und Soft Links sind richtig?

- ☐ Soft Links können nicht auf Verzeichnisse zeigen.
- ☐ Die Anzahl der Soft Links, die auf eine Datei verweisen, kann man mit Hilfe des Kommandos `ls -l` ermitteln.
- ☐ Beim Verschieben einer Datei, auf die ein Soft Link verweist, muss dieser manuell angepasst werden.
- ☐ Hard Links können nur innerhalb eines Dateisystems (Volume) eingesetzt werden.

Aufgabe 4: Welche Behauptungen zu Prozesskennungen (Prozess-IDs) sind gültig?

- ☐ Prozess-IDs werden in den jeweiligen i-Knoten gespeichert.
- ☐ Es ist sichergestellt, dass keine zwei gleichzeitig laufenden Prozesse die gleiche Prozess-ID haben.
- ☐ Die Prozess-IDs sind nur bzgl. eines Benutzers eindeutig.
- ☐ Die Prozess-ID der interaktiven Shell lässt sich mit dem Befehl `echo $$` ermitteln.
- ☐ Shell und Unter-Shell besitzen dieselbe Prozess-ID.
- ☐ Jeder Prozess besitzt eine wirkliche und eine effektive Prozess-ID (RPID und EPID).
- ☐ Der Prozess `init` hat stets die Prozess-ID `1`.
- ☐ Nach dem Systemaufruf `fork` besitzen Vater und Sohn dieselbe Prozess-ID.
- ☐ Die Prozess-ID eines Prozesses lässt sich mit dem Kommando `ps` ermitteln.

Aufgabe 5: Welche Aussagen sind richtig? Wird in der Shell interaktiv ein Kommando aufgerufen, für das ein Unterprozess erzeugt wird, dann

- ☐ verzweigt sich zuerst die Shell (`fork`).
- ☐ hat der Unterprozess und alle von ihm erzeugten Unter-Unterprozesse immer die Prozess-ID 0.
- ☐ wartet die Shell auf die Beendigung des Kommandos mittels `wait`.
- ☐ erhält die Shell anschließend den Beendigungsstatus (exit status) des Kommandos zurück.

Aufgabe 6: Die Shell (z. B. Bourne Shell oder bash) besticht durch

- ☐ ein ausgefeiltes objekt-orientiertes Typ- und Konstruktorkonzept.
- ☐ ein konsistentes Textsubstitutionskonzept.
- ☐ ein komplexes Synchronisierungskonzept zur Steuerung paralleler Prozesse.
- ☐ ein komfortables Variablen- und Konstanten-Repository mit Versionsverwaltung.
- ☐ direkte Anbindung an die Registry.

Aufgabe 7: Das Kommando `ls -l myScript` liefere die folgende Ausgabe:

```
-rwxr--r--    1 kai      students1       20 2004-05-03 16:35 myScript
```

Wie kann man allen Mitgliedern der Gruppe `students1` das Ausführungs- und Schreibrecht für die Datei `myScript` erteilen und die anderen Rechte beibehalten?

- ☐ `chmod 774 myScript`
- ☐ `chmod students1=read&write&execute myScript`
- ☐ `chmod g+rx myScript`
- ☐ `chmod 177 myScript`

Aufgabe 8: Welche der unten stehenden Aussagen bzgl. der Kommandointerpretation durch die Shell sind gültig?

- ☐ Durch Anhängen eines Prozentzeichens `%` an das Ende eines Kommandos wird dieses im Hintergrund ausgeführt.
- ☐ Kommandos, die in runden Klammern stehen, werden durch eine Untershell ausgeführt.
- ☐ Zwei Kommandos, die durch ein Semikolon voneinander getrennt sind, werden parallel abgearbeitet.
- ☐ Bei Verwendung einer Pipeline wartet die Shell auf die Beendigung des letzten Kommandos in der Pipe.
- ☐ Kommandos, die mit dem Ampersand-Zeichen `&` abgeschlossen werden, bekommen als Standardeingabe `/dev/null` zugewiesen, wenn nichts anderes angegeben ist.

Aufgabe 9: Betrachten Sie das folgende Skript:

```
cmd=$1
set `$cmd`
echo $1
```

Das Skript befinde sich im Verzeichnis `/home/staff/fix` und werde dort mit dem Parameter `pwd` aufgerufen. Welche Ausgabe erhalten Sie?

Aufgabe 10: In einem Shell-Skript soll die Standardausgabe und die Standardfehlerausgabe eines Kommandos unterdrückt werden. Was muss dazu an das Kommando angehangen werden?

- ☐ `1&2>/dev/null`
- ☐ `>/dev/null 2>&1`
- ☐ `2>/dev/null >/dev/null`
- ☐ `>/dev/null &2>1`

Aufgabe 11: Im aktuellen Verzeichnis befinde sich nur die Datei `myFile`. Diese soll in `yourFile` umbenannt werden. Welche der folgenden Kommandozeilen können dazu verwendet werden?

- ☐ `ln myFile yourFile; rm myFile`
- ☐ `rm myFile; cp myFile yourFile`
- ☐ `ren myFile yourFile`
- ☐ `mv myFile yourFile`

Aufgabe 12: Ein UNIX-Benutzer verwendet den Editor `ed`, um mit dem folgenden Befehl die Datei `myText` zu ändern:

```
1,$s/ä/ae/g
```

Ein Kollege gibt ihm den Tipp, dass man den gleichen Effekt auf der Kommandozeile mit dem Befehl

```
cat myText | tr 'ä' 'ae'
```

erreichen kann. Was meinen Sie? Begründen Sie Ihre Antwort!

Aufgabe 13: Zentraler Bestandteil des UNIX-Sicherheitskonzepts ist das so genannte Set-User-Id-Bit (kurz: s-Bit). Welche der folgenden Behauptungen sind korrekt?

- ☐ Nur der Super-User kann das s-Bit für eine Datei setzen.
- ☐ Bei der Passwort-Datei `/etc/passwd` ist das s-Bit gesetzt.
- ☐ Mit dem Kommando `chmod u+s myFile` wird das s-Bit für die Datei `myFile` gesetzt.
- ☐ Das s-Bit wird bei Shell-Skripten nicht beachtet.
- ☐ Kommandos mit gesetztem s-Bit dürfen nur vom Super-User ausgeführt werden.

Aufgabe 14: In der Shell seien die Variablen a und b wie folgt definiert:

```
a=echo
b="hallo $a"
```

Was liefern dann die nachstehenden Kommandos als Ausgabe?

a) `echo "$a ; $b"`

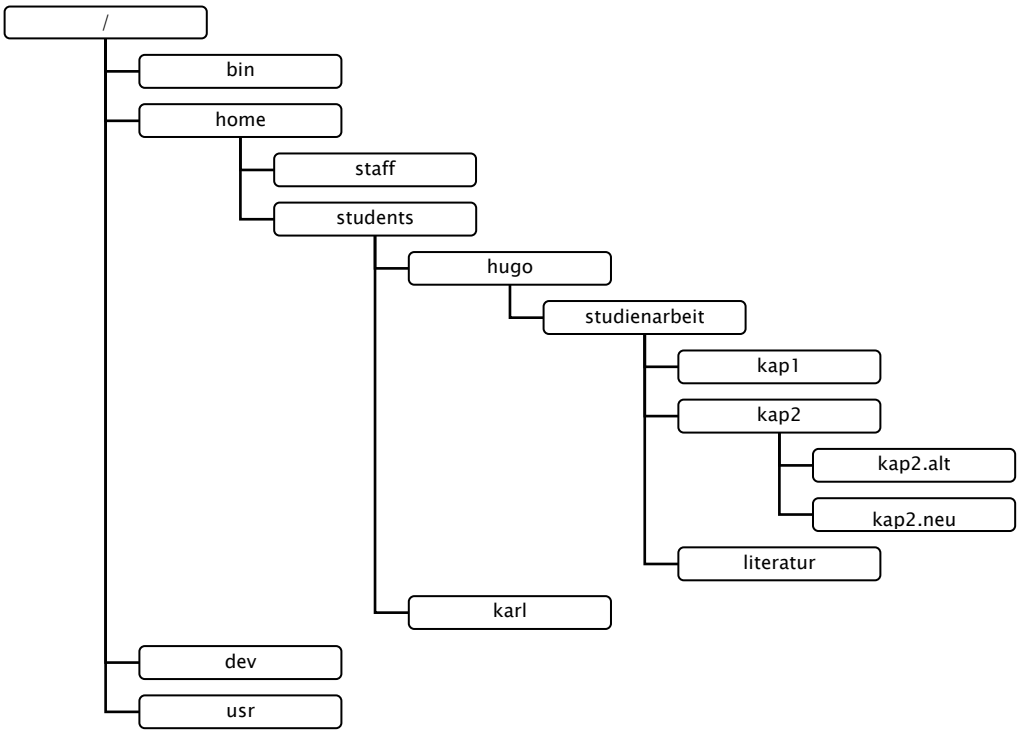
---

b) `echo $b ; $a`

---

c) `echo '$b ; $a'`

Aufgabe 15: Betrachten Sie den folgenden Ausschnitt eines Dateibaumes:



Nach dem Anmelden (Login) am System gibt hugo das Kommando

```
cd studienarbeit/kap2
```

ein.

a) Wie lautet der volle (absolute) Pfadname seines Heimatverzeichnisses?

---

b) Wie lautet der volle (absolute) Pfadname seines Arbeitsverzeichnisses?

---

c) hugo möchte in das Verzeichnis

```
/home/students/hugo/studienarbeit/kap1
```

wechseln. Wie lautet der dazu nötige relative Pfadname, mit dem hugo das Kommando `cd` aufrufen muss?

---

d) hugo führt jetzt das Kommando `cd ; pwd` aus. Wie lautet die Ausgabe?

Aufgabe 16: Das Shell-Skript `doubleEcho` soll jedes übergebene Argument doppelt ausgeben. Beispiel:

```
fix@holle:~> doubleEcho der Berg ruft
der der Berg Berg ruft ruft
fix@holle:~>
```

Ergänzen Sie die Lücken in dem folgenden Skript!

```
# Shell-Skript doubleEcho
while test ___________ -ne 0
do
   echo -n ___________
   ___________
done
echo
```

Aufgabe 17: Schreiben Sie ein Shell-Skript `doubleEcho2`, das die gleiche Funktion wie `doubleEcho` aus Aufgabe 16 hat. Verwenden Sie jedoch statt der `while`-Schleife eine `for`-Schleife!

Dipl.-Ing. Nabil Benamar
Dipl.-Math. Kai Schweinsberg

22. Juli 2004

Introduction to UNIX
Written Exam

Name:..... Surname:.....

Matr. No.:..... Enrolled Studies:.....

- Please answer all questions!
- Multiple choice questions may have more than one correct choice.
- This is a closed book exam!
- In case you need additional space to answer the question, please use the back of the sheets. Separate sheets of paper are not permitted.

Question	Points achievable	Points given
1	2,0	
2	4,0	
3	2,0	
4	4,5	
5	2,0	
6	2,0	
7	2,0	
8	2,5	
9	2,5	
10	2,0	
11	2,0	
12	3,0	
13	2,5	
14	3,0	
15	3,0	
16	4,0	
17	4,0	
Summe	**47,0**	

Viel Erfolg!

Question 1: What is a „T-fitting“ („T-Stück“) ?

- ☐ Having two Hard Links point to the same directory.
- ☐ Using several terminals on a single computer (older UNIX-Versions only).
- ☐ Using the `tee` command within a pipe.
- ☐ Passing the value of all shell variables to an undershell by means of the `make_t` command.

Question 2: Which of the following statements concerning the password file `/etc/passwd` are correct?

- ☐ Each user may list his own entry in the password file, the entries of all other users are hidden for security reasons.
- ☐ Only the super user may read the file.
- ☐ Any system user may read the file.
- ☐ A user may edit his or her password by editing the file (as an alternative to the `passwd` command).
- ☐ In the file each user has a home directory assigned to him or her.
- ☐ The password file determines the groups which individual users belong to.
- ☐ For security reasons, current UNIX-versions do not store passwords in the password file anymore.
- ☐ By adding a point to the end of an entry, the respective working directory is included in the search path of a user.

Question 3: Which statements concerning hard and soft links are correct?

- ☐ Soft links cannot point to directories.
- ☐ The number of soft links which point to a file can be detected using the `ls -l` command.
- ☐ When moving a file which has a soft link pointing to it, this link must be manually adjusted.
- ☐ Hard links may only be used within a single file system (volume).

Question 4: Which statements concerning process IDs are valid?

- ☐ Process IDs are stored in the respective i-nodes.
- ☐ It is assured that no two concurrently running processes have the same process ID.
- ☐ The process IDs are only unique with respect to each user.
- ☐ The process ID of the interactive shell may be returned with the command `echo $$`.
- ☐ Shell and Subshell have the same process ID.
- ☐ Each process owns a real and an effective process ID (RPID and EPID).
- ☐ The `init` process is always assigned process ID 1.
- ☐ Following a `fork` system call, father and son have the same process ID.
- ☐ The process ID of a process can be seen using the `ps` command.

Question 5: Which of the following statements is/are correct? When a command is started interactively by a shell and a subprocess is created for it, then

- ☐ the shell first creates another shell (`fork`).
- ☐ the subprocess and alle sub-subprocesses created by it are always assigned process ID 0.
- ☐ the shell waits for the termination of the command by means of `wait`.
- ☐ the shell receives the exit status of the finished command.

Question 6: The distinguishing features of the shell (e.g. Bourne Shell or bash) are:

- ☐ a highly refined object-oriented types and constructors concept.
- ☐ a consistent text substitution concept.
- ☐ a complex synchronization concept to control parallel processes.
- ☐ a comfortable repository to store variables and constants including version management.
- ☐ direct linking to the registry

Question 7: The command line `ls -l myScript` returns:

```
-rwxr--r--    1 kai      students1       20 2004-05-03 16:35 myScript
```

How can all members of the group `students1` be granted the rights to execute and write the file `myScript` while maintaining the other rights?

- ☐ `chmod 774 myScript`
- ☐ `chmod students1=read&write&execute myScript`
- ☐ `chmod g+rx myScript`
- ☐ `chmod 177 myScript`

Question 8: Which of the following statements concerning the command interpretation by the shell are correct?

- ☐ By attaching a percent character `%` to the end of a command, the command is executed in the background.
- ☐ Commands which are enclosed in round brackets, are executed by a subshell.
- ☐ Two commands which are separated by a semicolon are executed in parallel.
- ☐ When using a pipe, the shell waits for the termination of the last command in the pipe.
- ☐ Commands which have the apersand-sign `&` attached, are assigned `/dev/null` as standard input unless something else has been specified.

Question 9: Note the following script:

```
cmd=$1
set `$cmd`
echo $1
```

The script is in directory `/home/staff/fix` and is called there with the word `pwd` as parameter. Which output is produced?

Question 10: In a shell script both standard output and standard error output of a command are to be suppressed. What needs to be attached to the command?

- ☐ `1&2>/dev/null`
- ☐ `>/dev/null 2>&1`
- ☐ `2>/dev/null >/dev/null`
- ☐ `>/dev/null &2>1`

Question 11: The current directory contains only one file named `myFile`. This file is to be renamed `yourFile`. Which of the following command lines may be used?

- ☐ `ln myFile yourFile; rm myFile`
- ☐ `rm myFile; cp myFile yourFile`
- ☐ `ren myFile yourFile`
- ☐ `mv myFile yourFile`

Question 12: A UNIX-user calls the editor `ed` to change the file `myText` with the following command:

```
1,$s/ä/ae/g
```

A friend gives him a hint that the same effect can be achieved with the command line

```
cat myText | tr 'ä' 'ae'
```

What is your opinion? Substantiate your answer!

Question 13: A central component of the UNIX security concept is the so-called Set-User-Id-Bit (s-Bit for short). Which of the following statements is/are correct?

- ☐ Only the Super-User may set the s-Bit for a file.
- ☐ The password file `/etc/passwd` has the s-Bit set.
- ☐ The command `chmod u+s myFile` sets the s-Bit for `myFile`.
- ☐ For shell scripts, the s-Bit is ignored.
- ☐ Commands which have the s-Bit set may only be invoked by the Super-User.

Question 14: In the shell variables a and b are defined as follows:

```
a=echo
b="hallo $a"
```

What is returned by the commands below?

a) `echo "$a ; $b"`

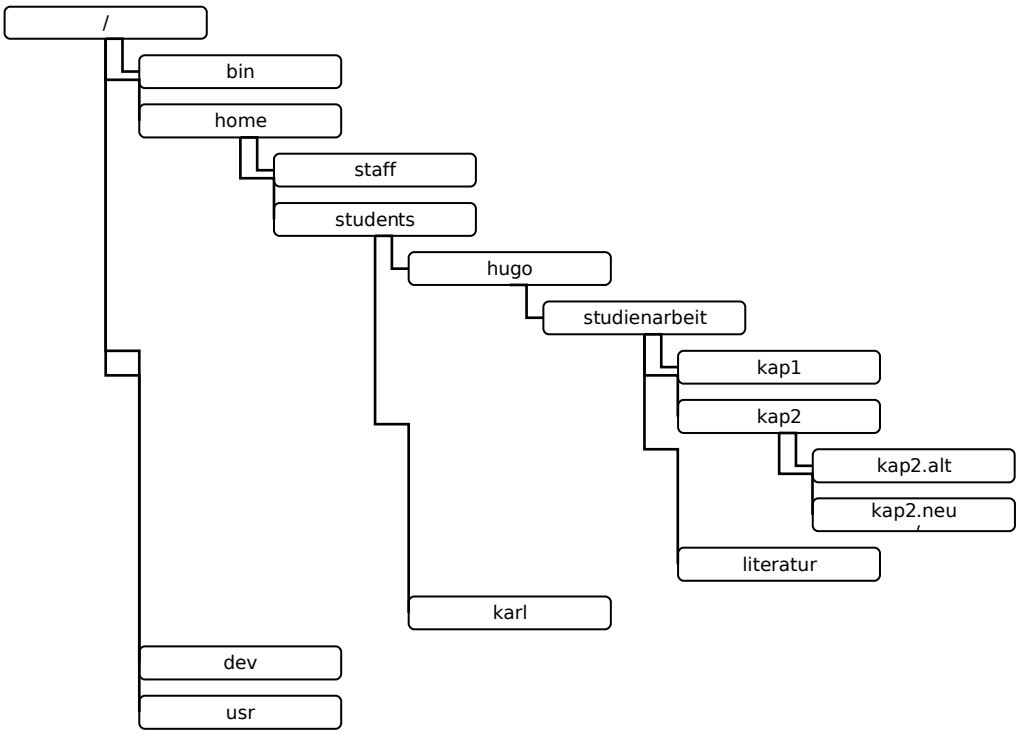
---

b) `echo $b ; $a`

---

c) `echo '$b ; $a'`

Question 15: Consider the following part of a file tree:



Following his login to the system, `hugo` enters the command

```
cd studienarbeit/kap2
```

a) What is the complete (absolute) pathname of his home directory?

---

b) What is the complete (absolute) pathname of his working directory?

---

c) `hugo` wants to change into directory

```
/home/students/hugo/studienarbeit/kap1
```

What is the relative pathname which `hugo` must use in the `cd` command?

---

d) `hugo` invokes the command line `cd ; pwd`. What is the resulting output?

Question 16: The shell script `doubleEcho` is to double each given argument into the output. As an example, consider:

```
fix@holle:~> doubleEcho der Berg ruft
der der Berg Berg ruft ruft
fix@holle:~>
```

Fill in the gaps in the following script!

```
# Shell-Script doubleEcho
while test __________ -ne 0
do
   echo -n __________
   __________
done
```

Question 17: Write a shell script `doubleEcho2` which has the same function as the shell script `doubleEcho` from Question 16. However, use a `for`-loop instead of the `while`-loop!

Dipl.-Ing. Nabil Benamar
Dipl.-Math. Kai Schweinsberg

22. Juli 2004

Einführung in UNIX

Klausur

Name: ..... Vorname: .....

Matr. Nr.: ..... Fachbereich: .....

- Bearbeiten Sie alle Fragen!
- Bei Ankreuzaufgaben muss mindestens eine Antwort angekreuzt werden. Sind alle Antworten angekreuzt, so wird die Aufgabe nicht gewertet.
- Hilfsmittel sind nicht zugelassen!
- Falls Sie für die Beantwortung der Aufgaben zusätzlichen Platz benötigen, verwenden Sie die Rückseiten. Separate Blätter sind nicht zugelassen.

Aufgabe	Mögliche Punkte	Erreichte Punkte
1	2,0	2,0
2	4,0	4,0
3	2,0	2,0
4	4,5	4,5
5	2,0	2,0
6	2,5	2,5
7	2,0	2,0
8	2,5	2,5
9	3,0	3,0
10	2,0	2,0
11	2,0	2,0
12	3,0	3,0
13	2,5	2,5
14	3,0	3,0
15	3,0	3,0
16	4,0	4,0
17	4,0	4,0
Summe	**48,0**	**48,0**

Aufgabe 1: Was versteht man unter einem „T-Stück“?

- ☐ Das Setzen zweier Hard Links auf dasselbe Verzeichnis.
- ☐ Den Einsatz mehrerer Terminals an einem Rechner (nur in älteren UNIX-Versionen).
- ☒ Das Verwenden des Kommandos `tee` innerhalb einer Pipeline.
- ☐ Die Übergabe der Werte aller Shell-Variablen an eine Untershell durch das Kommando `make_t`.

Aufgabe 2: Welche der folgenden Aussagen zur Passwort-Datei `/etc/passwd` sind richtig?

- ☐ Jeder Benutzer kann nur seinen eigenen Eintrag in der Passwort-Datei auflisten, die Einträge der anderen Benutzer bleiben aus Sicherheitsgründen verborgen.
- ☐ Nur der Super-User kann die Datei lesen.
- ☒ Jeder Benutzer des Systems kann die Datei lesen.
- ☐ Jeder Benutzer kann sein Passwort durch Editieren der Datei ändern (Alternative zum Kommando `passwd`).
- ☒ In der Datei wird jedem Benutzer ein Heimatverzeichnis zugeordnet.
- ☐ Die Passwort-Datei legt fest, welchen Gruppen die einzelnen Benutzer angehören.
- ☒ Aktuelle UNIX-Versionen speichern in der Passwort-Datei aus Sicherheitsgründen keine Passwörter mehr.
- ☐ Durch das Hinzufügen eines Punkts am Ende eines Eintrags kann festgelegt werden, dass das jeweilige Arbeitsverzeichnis in den Suchpfad des Benutzers einbezogen wird.

Aufgabe 3: Welche Aussagen zu Hard und Soft Links sind richtig?

- ☐ Soft Links können nicht auf Verzeichnisse zeigen.
- ☐ Die Anzahl der Soft Links, die auf eine Datei verweisen, kann man mit Hilfe des Kommandos `ls -l` ermitteln.
- ☒ Beim Verschieben einer Datei, auf die ein Soft Link verweist, muss dieser manuell angepasst werden.
- ☒ Hard Links können nur innerhalb eines Dateisystems (Volume) eingesetzt werden.

Aufgabe 4: Welche Behauptungen zu Prozesskennungen (Prozess-IDs) sind gültig?

- ☐ Prozess-IDs werden in den jeweiligen i-Knoten gespeichert.
- ☒ Es ist sichergestellt, dass keine zwei gleichzeitig laufenden Prozesse die gleiche Prozess-ID haben.
- ☐ Die Prozess-IDs sind nur bzgl. eines Benutzers eindeutig.
- ☒ Die Prozess-ID der interaktiven Shell lässt sich mit dem Befehl `echo $$` ermitteln.
- ☐ Shell und Unter-Shell besitzen dieselbe Prozess-ID.
- ☐ Jeder Prozess besitzt eine wirkliche und eine effektive Prozess-ID (RPID und EPID).
- ☒ Der Prozess `init` hat stets die Prozess-ID 1.
- ☐ Nach dem Systemaufruf `fork` besitzen Vater und Sohn dieselbe Prozess-ID.
- ☒ Die Prozess-ID eines Prozesses lässt sich mit dem Kommando `ps` ermitteln.

Aufgabe 5: Welche Aussagen sind richtig? Wird in der Shell interaktiv ein Kommando aufgerufen, für das ein Unterprozess erzeugt wird, dann

- ☒ verzweigt sich zuerst die Shell (`fork`).
- ☐ hat der Unterprozess und alle von ihm erzeugten Unter-Unterprozesse immer die Prozess-ID 0.
- ☒ wartet die Shell auf die Beendigung des Kommandos mittels `wait`.
- ☒ erhält die Shell anschließend den Beendigungsstatus (exit status) des Kommandos zurück.

Aufgabe 6: Die Shell (z. B. Bourne Shell oder bash) besticht durch

- ☐ ein ausgefeiltes objekt-orientiertes Typ- und Konstruktorkonzept.
- ☒ ein konsistentes Textsubstitutionskonzept.
- ☐ ein komplexes Synchronisierungskonzept zur Steuerung paralleler Prozesse.
- ☐ ein komfortables Variablen- und Konstanten-Repository mit Versionsverwaltung.
- ☐ direkte Anbindung an die Registry.

Aufgabe 7: Das Kommando `ls -l myScript` liefere die folgende Ausgabe:

```
-rwxr--r--    1 kai      students1       20 2004-05-03 16:35 myScript
```

Wie kann man allen Mitgliedern der Gruppe `students1` das Ausführungs- und Schreibrecht für die Datei `myScript` erteilen und die anderen Rechte beibehalten?

- ☒ `chmod 774 myScript`
- ☐ `chmod students1=read&write&execute myScript`
- ☐ `chmod g+rx myScript`
- ☐ `chmod 177 myScript`

Aufgabe 8: Welche der unten stehenden Aussagen bzgl. der Kommandointerpretation durch die Shell sind gültig?

- ☐ Durch Anhängen eines Prozentzeichens `%` an das Ende eines Kommandos wird dieses im Hintergrund ausgeführt.
- ☒ Kommandos, die in runden Klammern stehen, werden durch eine Untershell ausgeführt.
- ☐ Zwei Kommandos, die durch ein Semikolon voneinander getrennt sind, werden parallel abgearbeitet.
- ☒ Bei Verwendung einer Pipeline wartet die Shell auf die Beendigung des letzten Kommandos in der Pipe.
- ☒ Kommandos, die mit dem Ampersand-Zeichen `&` abgeschlossen werden, bekommen als Standardeingabe `/dev/null` zugewiesen, wenn nichts anderes angegeben ist.

Aufgabe 9: Betrachten Sie das folgende Skript:

```
cmd=$1
set `$cmd`
echo $1
```

Das Skript befinde sich im Verzeichnis `/home/staff/fix` und werde dort mit dem Parameter `pwd` aufgerufen. Welche Ausgabe erhalten Sie?

/home/staff/fix

Aufgabe 10: In einem Shell-Skript soll die Standardausgabe und die Standardfehlerausgabe eines Kommandos unterdrückt werden. Was muss dazu an das Kommando angehangen werden?

- ☐ `1&2>/dev/null`
- ☒ `>/dev/null 2>&1`
- ☒ `2>/dev/null >/dev/null`
- ☐ `>/dev/null &2>1`

Aufgabe 11: Im aktuellen Verzeichnis befinde sich nur die Datei `myFile`. Diese soll in `yourFile` umbenannt werden. Welche der folgenden Kommandozeilen können dazu verwendet werden?

- ☒ `ln myFile yourFile; rm myFile`
- ☐ `rm myFile; cp myFile yourFile`
- ☐ `ren myFile yourFile`
- ☒ `mv myFile yourFile`

Aufgabe 12: Ein UNIX-Benutzer verwendet den Editor `ed`, um mit dem folgenden Befehl die Datei `myText` zu ändern:

```
1,$s/ä/ae/g
```

Ein Kollege gibt ihm den Tipp, dass man den gleichen Effekt auf der Kommandozeile mit dem Befehl

```
cat myText | tr 'ä' 'ae'
```

erreichen kann. Was meinen Sie? Begründen Sie Ihre Antwort!

Die beiden Befehle sind nicht gleichwertig! Der erste Befehl ersetzt jedes in der Datei „myText" vorkommende „ä" durch die Zeichenfolge „ae". Der zweite Befehl führt jedoch nur eine buchstabenweise Ersetzung durch: „ä" wird durch „a" ersetzt!

Anmerkung: Es wurden ebenfalls Punkte auf die Feststellung vergeben, dass der erste Befehl es ermöglicht, die Änderungen dauerhaft zu speichern, wohingegen der zweite Befehl nur eine Ausgabe des geänderten Textes liefert.

Aufgabe 13: Zentraler Bestandteil des UNIX-Sicherheitskonzepts ist das so genannte Set-User-Id-Bit (kurz: s-Bit). Welche der folgenden Behauptungen sind korrekt?

- ☐ Nur der Super-User kann das s-Bit für eine Datei setzen.
- ☐ Bei der Passwort-Datei `/etc/passwd` ist das s-Bit gesetzt.
- ☒ Mit dem Kommando `chmod u+s myFile` wird das s-Bit für die Datei `myFile` gesetzt.
- ☒ Das s-Bit wird bei Shell-Skripten nicht beachtet.
- ☐ Kommandos mit gesetztem s-Bit dürfen nur vom Super-User ausgeführt werden.

Aufgabe 14: In der Shell seien die Variablen `a` und `b` wie folgt definiert:

```
a=echo
b="hallo $a"
```

Was liefern dann die nachstehenden Kommandos als Ausgabe?

a) `echo "$a ; $b"`

echo ; hallo echo

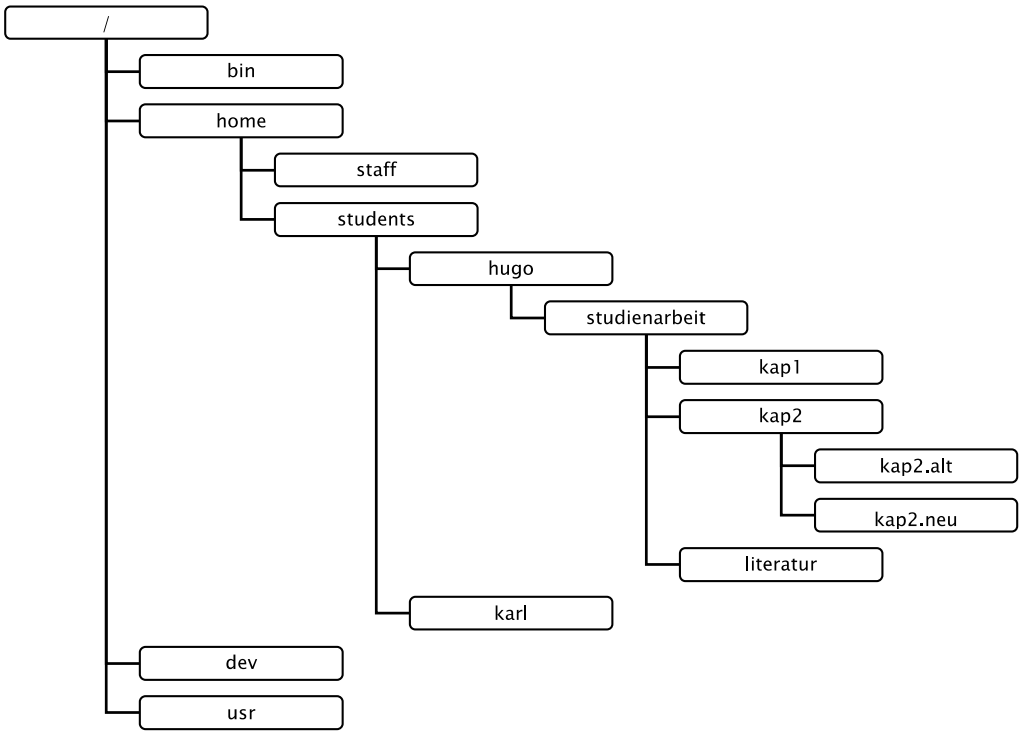
b) `echo $b ; $a`

hallo echo (und eine leere Zeile)

c) `echo '$b ; $a'`

$b ; $a

Aufgabe 15: Betrachten Sie den folgenden Ausschnitt eines Dateibaumes:



Nach dem Anmelden (Login) am System gibt `hugo` das Kommando

```
cd studienarbeit/kap2
```

ein.

a) Wie lautet der volle (absolute) Pfadname seines Heimatverzeichnisses?

/home/students/hugo

b) Wie lautet der volle (absolute) Pfadname seines Arbeitsverzeichnisses?

/home/students/hugo/studienarbeit/kap2

c) `hugo` möchte in das Verzeichnis

```
/home/students/hugo/studienarbeit/kap1
```

wechseln. Wie lautet der dazu nötige relative Pfadname, mit dem `hugo` das Kommando `cd` aufrufen muss?

../kap1

d) `hugo` führt jetzt das Kommando `cd ; pwd` aus. Wie lautet die Ausgabe?

/home/students/hugo

Aufgabe 16: Das Shell-Skript `doubleEcho` soll jedes übergebene Argument doppelt ausgeben. Beispiel:

```
fix@holle:~> doubleEcho der Berg ruft
der der Berg Berg ruft ruft
fix@holle:~>
```

Ergänzen Sie die Lücken in dem folgenden Skript!

```
# Shell-Skript doubleEcho
while test ___$#___ -ne 0
do
   echo -n "$1 $1 "
   _shift___
done
echo
```

Aufgabe 17: Schreiben Sie ein Shell-Skript `doubleEcho2`, das die gleiche Funktion wie `doubleEcho` aus Aufgabe 16 hat. Verwenden Sie jedoch statt der `while`-Schleife eine `for`-Schleife!

```
for i
do
   echo -n "$i $i "
done
echo
```

Prof. Dr. Lutz Wegner
Dipl.-Ing. Nabil Benamar

22.10.2004

Klausur zur Lehrveranstaltung „UNIX“ vom WS 2004/05

Name: ..... Vorname: .....

Matr.Nr.: ..... FB: .....

Bearbeiten Sie alle Fragen! Bei Ankreuzaufgaben kann mehr als eine Antwort richtig sein. Hilfsmittel sind nicht zugelassen.

Aufgabe	Punkte max.	Punkte erreicht
1	2	
2	2	
3	2	
4	2	
5	2	
6	2	
7	2	
8	2	
9	2	
10	2	
11	2	
12	2	
13	2	
14	2	
15	2	
Summe	**30**	

Aufgabe 1:

Im Kurs üben wir mit einem SSH-Fenster unter Windows, das eine sichere Verbindung des PC als Client zu einem UNIX-Server herstellt. Was ist richtig?

(X) Zum Verbinden meldet man sich am Server mit dem normalen login-Paßwort an.

(X) Auch dieses Pseudoterminal ist beim Server als Gerät in `/dev` eingetragen.

( ) Man bekommt allerdings dann keine Bourne-Shell oder `bash`, sondern nur die `ssh`.

( ) Nur über diesen Umweg unterstützt UNIX mehrere Terminals an einem Rechner.

Aufgabe 2:

Ein Anwender kann eigene Kommandos als Shell-Skripte realisieren. Die mit der UNIX-Installation gelieferten Kommandos sind dagegen binaries.

(X) Ja, jedenfalls die meisten, schon aus Effizienzgründen.

( ) Ja, alle grundsätzlich, weil es nicht anders geht (SUID-Bit).

( ) Ja, außer das SUID-Bit ist gesetzt.

( ) Nein, alle Kommandos sind Shell-Skripte, sie wirken nur nach außen wie binaries.

Aufgabe 3:

Der Aufruf `ls | wc -l` liefere die Zahl 13. Was liefert der Aufruf `echo 'ls' | wc -l`

1

Aufgabe 4:

Wird ein BUILTIN-Kommando wie `cd` oder `eval` ausgeführt, überlagert sich dann immer die Shell, ohne einen neuen Prozeß zu starten?

( ) Ja.

(X) Nein.

( ) Kommt darauf an, ob das Kommando mit einem Punkt beginnt.

( ) Kommt darauf an, ob die Shell interaktiv ist und welche Shell es ist (`bash` oder `sh`).

Aufgabe 5:

Wächst eine Normaldatei und paßt sie nicht mehr in den alten Datenblock auf der Platte, wird dann ein neuer *inode* für sie angelegt und kann man das mit `ls -i` beobachten?

( ) Ja, immer.

( ) Ja, aber nur wenn die Datei ein `hard link` war.

(X) Nein, *inode* bleibt gleich.

( ) Implementationsabhängig.

( ) Frage macht keinen Sinn, weil Normaldateien keine *inodes* haben, nur Verzeichnisse.

Aufgabe 6:

Mit der Option `--` kann man die weiter rechts stehenden Optionen „ausschalten“, d.h. sie werden als normale Argumente genommen. Wie lautet der fehlende Teil in `ls` für die Ausgabe unten?

```
wegner@holle:~> ls -d -- -l .
/bin/ls: -l: Datei oder Verzeichnis nicht gefunden
.
wegner@holle:~>
```

Aufgabe 7:

Prof. Fix möchte die Belegung von `$PATH` ändern, speziell Teile streichen. Dazu hat er den Wert mit `echo $PATH >editpath` in `editpath` geschrieben und `editpath` mit seinem Lieblingseditor bearbeitet, z.B. mit `vi`. Jetzt soll der so geänderte Wert `PATH` wieder zugewiesen werden. Das geht z.B. mit Kommandosubstitution. Ergänzen Sie den fehlenden Teil.

```
PATH='__cat editpath_________'
```

Aufgabe 8:

Im Skript wird ein kleines Shell-Skript `cx` vorgestellt als Abkürzung für das häufig gebrauchte `chmod +x`. Ergänzen Sie die fehlende Stelle!

```
wegner@holle:~> echo 'chmod +x $*' >cx
wegner@holle:~> cx cx
-bash: ./cx: Keine Berechtigung
wegner@holle:~> ____sh____ cx cx
wegner@holle:~>
```

Aufgabe 9:

Mit dem `wall`-Kommando kann man allen zur Zeit angemeldeten Teilnehmern, die `write` zulassen, also nicht `mesg n` gesetzt haben, eine Botschaft schicken. Mit `who` bekommt man die angemeldeten Teilnehmer heraus, mit der Option `-q` (query) nur deren login-Namen und eine blöde Zählung `# Benutzer=`*n* am Schluß, die man aber mit `head -n -1` (Ausgabe ohne letzte Zeile) wegbekommt.

Das Shell-Skript `mywall` soll `wall` nachempfunden sein und bei Aufruf mit `mywall botschaft` den Inhalt der Datei `botschaft` an die angemeldeten Teilnehmer verschicken. Ergänzen Sie die fehlenden Teile.

mywall

```
message=$1
set `who -q | head -n -1`
for user
do
  write $user <$message
done
```

Aufgabe 10:

Man möchte `mywall` aus Aufgabe 9 zu einem späteren Zeitpunkt starten. Das kann man mit dem Kommando

(X) `at`

( ) `nohup`

( ) `cronicle`

( ) `schedule`

( ) `run by`

Aufgabe 11:

Wird mit `ln` ein harter Link auf eine Datei eingetragen, ist dieser neue Alias-Name gleichberechtigt mit dem alten Namen. Nehmen wir an, wir vergeben nochmal einen Namen mit `ln`, diesmal aber auf den Alias-Namen. Ist dieser „Alias-Alias“ auch wieder gleichberechtigt?

(X) Ja.

( ) Nein.

( ) Kann nicht passieren, weil `ln` das verbietet.

( ) Hängt vom Besitzer ab (alle Alias-Namen gleichberechtigt für `root`, sonst nicht).

( ) Hängt von den Dateirechten ab.

Aufgabe 12:

Das Textsegment eines Prozesses läßt sich von mehreren Prozessen gleichzeitig nutzen.

(X) Ja, wenn der Code READ ONLY ist.

( ) Ja, wenn die Mitbenutzerprozesse Kinder des Prozesses sind, der das Textsegment angelegt hat.

( ) Ja, bis zu dem Zeitpunkt, an dem der ursprüngliche Prozeß, der das Segment anlegte, noch arbeitet. Scheidet er aus, müssen auch alle anderen abgebrochen werden.

( ) Nein, nur die Datensegmente lassen sich gemeinsam nutzen.

( ) Nein, das verbieten die Sicherheitsregeln (getrennte Adreßräume).

Aufgabe 13:

Im Kurs wird bei `ls -l >tricky` beobachtet, daß `tricky` selbst seine Größe noch mit 0 angibt, hinterher `ls -l tricky` aber die „richtige“ Größe angibt. Das ist nicht immer so. Bei Eingabe von

```
ls -lR . /home >tricky 2>/dev/null
```

enthält `tricky` zwei unterschiedliche Ausgaben über sich selbst (`tricky` steht in `/home/staff/wegner`), nämlich

```
-rw-r--r--  1 wegner students1        0 2004-10-18 12:38 tricky
```

und

```
-rw-r--r--  1 wegner students1     3388 2004-10-18 12:38 tricky
```

Danach liefert `ls -l tricky` sogar die Größe 9195. Warum?

( ) Die Shell richtet in diesem Fall eben nicht die Umgebung ein.

( ) Das `ls`-Kommando weiß, daß es umgelenkt wird.

(X) Die Ausgabe von `ls` ist so groß, daß Teile davon schon in `tricky` stehen, wenn `ls` zum zweiten Mal das Verzeichnis von `tricky` auflistet.

( ) Hängt damit zusammen, daß zwei Verzeichnisse durchsucht werden (Angabe `2>/dev/null`).

Aufgabe 14:

Die Beschreibung des write-Kommandos lautet wie folgt.

```
WRITE(1)            Linux Programmer's Manual            WRITE(1)

NAME
     write - send a message to another user
SYNOPSIS
     write user [ttyname]
DESCRIPTION
     Write allows you to communicate with other users, by copy-
```

```
ing lines from your terminal to theirs.
...
```

Betrachten Sie den Dialog unten (Annahme: `wegner` ist an `pts/5` angemeldet).

```
wegner@holle:~> write pts/5
write: Ihr Schreibzugriff ist abgeschaltet.
wegner@holle:~> mesg y
wegner@holle:~> write pts/5
write: pts/5 is not logged in
wegner@holle:~>
```

Was ist richtig?

(X) Linux scheint zuerst zu prüfen, ob Schreibzugriff auf das eigene Terminal besteht.

(X) Man kann zusätzlich angeben, auf welches offene Terminal eines Teilnehmers man schreiben will.

(X) Ohne Login-Namen, nur mit Terminalangabe, geht es nicht.

(X) `write` produziert offenbar Fehlermeldungen auf Deutsch und auf Englisch.

(X) Die Angaben `write wegner` und `write wegner pts/5` wären unter der Annahme oben beide korrekt.

Aufgabe 15:

Mit der Option `-U` liefert `ls` die Einträge unsortiert in der Folge, wie sie im Katalog nacheinander kommen. Nach Neuanlegen der Datei `geheim2` sei die Ausgabe wie folgt.

```
wegner@holle:~> ls -U
Documents    bin   klausur   geheim2  parentID  koenig  geheim
public_html  kurs  dreiargs  nn       foenig    mbox
wegner@holle:~> rm klausur
wegner@holle:~> touch wohin
wegner@holle:~> ls -U
????????
```

Wo taucht jetzt die neuangelegte Datei `wohin` in der Auflistung mit `ls -U` auf?

( ) Ganz vorne vor `Documents`.

( ) Ganz hinten nach `mbox`.

(X) Wo vorher `klausur` stand.

( ) Kann man nicht sagen, hängt vom Modifikationsdatum der anderen Dateien ab.

( ) Nach allen Verzeichnissen und vor allen anderen Normaldateien.

Prof. Dr. Lutz Wegner
Dipl.-Math. Kai Schweinsberg
Dipl.-Ing. Nabil Benamar

11.3.2005

Klausur zur Lehrveranstaltung „UNIX“ vom WS 2004/05

Musterlösung

Name: ..... Vorname: .....

Matr.-Nr.: ..... FB: .....

Bearbeiten Sie alle Fragen! Bei Ankreuzaufgaben kann mehr als eine Antwort richtig sein. Hilfsmittel sind nicht zugelassen.

Aufgabe	**Punkte max.**	**Punkte erreicht**
1	2	
2	2	
3	2	
4	2	
5	2	
6	2	
7	2	
8	2	
9	2	
10	3	
11	3	
12	2	
13	2	
14	6	
Summe	**34**	

Aufgabe 1:

Das Betriebssystem LINUX läßt sich heute auf ganz unterschiedlicher Hardware einsetzen, vom Server bis zum Notebook.

(X) richtig.

( ) falsch, nur Einzel-PC.

( ) falsch, nur Notebook.

( ) falsch, nicht auf Notebook.

Aufgabe 2:

Wenn im Zusammenhang mit den Kommandos oder der Shell von Argumenten gesprochen wird, meint man damit

(X) eine Folge von nicht-leeren Worten, die mit Blank, Tabulator oder Newline getrennt sind.

( ) eine Folge von Worten, die mit ASCII Null getrennt sind.

( ) Werte, die sich hinter Variablen wie $HOME, $PATH, $# verbergen.

( ) immer nur die Werte, die sich aus einer Kommandosubstitution ergeben.

Aufgabe 3:

Die Eingabeumlenkung regelt jedes Kommando individuell.

( ) Ja, alle.

( ) Ja, aber nur die eingebauten Kommandos.

(X) Nein, regelt die Shell.

( ) Nein, ist fest auf `/dev/null` eingestellt.

Aufgabe 4:

Im Editor `vi` kann man im Kommandomodus mit den Cursortasten navigieren, d.h. den Cursor an eine gewünschte Stelle setzen.

(X) Ja.

( ) Nein, geht nur im Eingabemodus.

( ) Nein, der `vi` ist zeilenorientiert.

( ) Ja, wenn die KDE-Oberfläche mit multiplen Fenstern aktiviert ist.

Aufgabe 5:

Einen harten Link auf eine Datei kann man nur eintragen, wenn man die i-Nummer (i-node) kennt, unter der die Datei mit dem Originalnamen ursprünglich eingetragen wurde.

( ) Richtig, kann man mit `ls -i` rausbekommen

( ) Richtig, deshalb ist der Softlink mit `ln -s` die Regel.

(X) Falsch, der Link kann über den Originalnamen oder einen der Aliasnamen eingetragen werden.

( ) Falsch, bei harten Links gibt es keine i-Nummern.

( ) Kann man so nicht beantworten, hängt vom Dateisystem ab (z.B. ob es ein NFS ist).

Aufgabe 6:

Nach einem Aufruf von `fork()` folgt in der Regel ein `exec()` zur Überlagerung des neuen Prozesses. Warum?

( ) Sonst hat der Sohnprozeß keine neue Prozeßnummer.

( ) Sonst kann der Vater kein `wait()` machen.

(X) Sonst hätten Vater und Sohn dasselbe Textsegment (was es manchmal gibt).

( ) Sonst würde der Sohn die Umgebung nicht erben.

( ) Sonst kann man nicht unterscheiden, wer Vater und Sohn ist.

Aufgabe 7:

Welche Ausgabe liefert die folgende Pipeline?

```
echo Hallo Herr Mueller | tr -s [A-z]
```

Ausgabe

Halo Her Mueler

Aufgabe 8:

Das `false`-Kommando unter LINUX macht alles falsch, das aber sehr effizient.

( ) Kann es nicht geben.

(X) OK, wenn es das gibt, dann liefert der Exit-Status einen Wert ungleich 0.

( ) OK, wenn es das gibt, dann liefert der Exit-Status 0.

(X) OK, wenn es das gibt, dann wird in `false || MeinKommando` immer `MeinKommando` ausgeführt.

Aufgabe 9:

Betrachten Sie den folgenden Aufruf und die zugehörige Ausgabe.

```
wegner@holle:~> date; sleep 5; date
Mo Mär  7 14:32:27 CET 2005
Mo Mär  7 14:32:32 CET 2005
```

Bei Aufruf mit

```
wegner@holle:~> echo `date` `sleep 5` `date`
```

erhält man

(X) die Ausgabe in einer Zeile.

( ) die Ausgabe in zwei Zeilen wie oben.

(X) die gesamte Ausgabe erst nach 5 Sekunden.

( ) eine Fehlermeldung.

( ) die Ausgabe `date sleep 5 date`.

Aufgabe 10:

Für ein leeres Verzeichnis liefert `ls -a` die Ausgabe

```
wegner@holle:~/kurs/aushang/leer> ls -a

.  ..

wegner@holle:~/kurs/aushang/leer>
```

Lenkt man die Ausgabe per Pipe aber in das `wc`-Kommando um, ergibt sich überraschenderweise `2  2  5`.

Erklären Sie die Werte!

ls gibt bei Verbindung mit Pipe einen Eintrag je Zeile aus, damit 2 Zeilen, 2 Worte („." und „.."), 5 Zeichen (3 Punkte und zwei Newlines).

Aufgabe 11:

Erläutern Sie die Unterschiede der beiden Ausgaben von `ps` unten.

```
wegner@holle:~> ps
PID TTY          TIME CMD
29500 pts/0    00:00:00 bash
32180 pts/0    00:00:00 ps
wegner@holle:~> (ps; date)
PID TTY          TIME CMD
29500 pts/0    00:00:00 bash
32183 pts/0    00:00:00 bash
32184 pts/0    00:00:00 ps
Di Mär  8 11:53:08 CET 2005
wegner@holle:~>
```

Im zweiten Aufruf werden ps und date in einer Untershell aufgerufen. Damit sind zum Zeitpunkt der Abarbeitung von ps zwei Shell-Prozesse vorhanden, date übrigens noch nicht.

Aufgabe 12:

Die Ausgaben von `ls` und von `echo *` unterscheiden sich - wie unten zu sehen - bis auf Formatierungen (die `ls` macht) kaum.

```
wegner@holle:~/kurs> ls
aushang  echo2  echo3  l2  lc  mkfile  testexit  writemail
wegner@holle:~/kurs> echo *
aushang echo2 echo3 l2 lc mkfile testexit writemail
```

Unter Beachtung der Tatsache, daß `aushang` ein Verzeichnis ist, liefert

```
wegner@holle:~/kurs> ls *
```

( ) eine bis auf Formatierungen gleiche Ausgabe wie zuvor.

(X) eine andere Ausgabe mit zusätzlich den Dateien von `aushang`, sofern vorhanden.

( ) eine Meldung der Art `Datei * nicht gefunden`

( ) eine andere Ausgabe, die nur die Dateien von `aushang` enthält, sofern vorhanden.

Aufgabe 13:

Der Link-Zähler für ein Verzeichnis steht auf 2, wenn in diesem Verzeichnis kein Unterverzeichnis vorkommt, auf 3 oder mehr, wenn es mindestens ein Unterverzeichnis gibt.

(X) Ja.

( ) Nein, steht auf 1, wenn in diesem Verzeichnis kein Unterverzeichnis vorkommt, auf 2 oder mehr, wenn es Unterverzeichnisse gibt.

( ) Kann man nicht so pauschal sagen, kommt auf die Anzahl der sonstigen Dateien an.

( ) Nein, steht immer auf 2.

( ) Nein, kommt auf die Anzahl der mit `ln` gesetzten harten Links auf dieses Verzeichnis an.

Aufgabe 14:

Betrachten Sie das folgende Shell-Skript `gelinkt`. Wie an der Ausgabe von `echo` zu erkennen, werden paarweise die Dateinamen aus der Argumentliste ausgegeben, die zur selben Datei gehören. Das Kommando `test` mit der Option `-ef` (equal file) prüft ob beide Ausdrücke Dateinamen mit gleicher *Device-Number* und gleichem *inode* sind. Der typische Aufruf wäre `gelinkt *` für alle Einträge im aktuellen Verzeichnis.

```
for i
do
    for j
    do
        if
            test  $i -ef $j  -a $i != $j
        then echo $i and $j haben gleichen i-node
        fi
    done
done
```

(a) Wie oft wird `test` aufgerufen, wenn *n* Argumente übergeben werden?

$O(n^2)$

(b) Wozu dient der mit `-a` (and) angefügte zweite Vergleich?

Damit nicht zwei gleiche Namen in der Ausgabe erscheinen, die ja trivialerweise denselben inode haben.

(c) Wird dank dieses Tests jedes Pärchen genau einmal ausgegeben?

Nein, jedes Paar zweimal.

(d) Welcher Zusammenhang besteht zur Ausgabe von `ls -i | sort` ?

Sortiert man die Ausgabe von ls -i (Auflisten mit i-Nummer), dann stehen Dateinamen, die auf die gleiche Datei verweisen, hintereinander, sind also leichter zu erkennen.

Prof. Dr. Lutz Wegner
Dipl.-Ing. Nabil Benamar 15. Juli 2005

Einführung in UNIX

Klausur zum Sommersemester 2005

Name: ..... Vorname: .....

Matr. Nr.:..... Fachbereich:.....

- Bearbeiten Sie alle Fragen!
- Bei Ankreuzaufgaben kann mehr als eine Antwort richtig sein.
- Hilfsmittel sind nicht zugelassen!
- Falls Sie für die Beantwortung der Aufgaben zusätzlichen Platz benötigen, verwenden Sie die Rückseiten. Separate Blätter sind nicht zugelassen.

Aufgabe	Mögliche Punkte	Erreichte Punkte
1	2	
2	3	
3	4	
4	4	
5	4	
6	2	
7	3	
8	3	
9	3	
10	2	
11	4	
12	4	
13	4	
14	4	
15	4	
Summe	50	

Aufgabe 1: Manche Dinge werden mit dem Alter immer besser. Das gilt auch für UNIX. Was gilt in allen Versionen und Varianten (einschließlich LINUX), wie in der Vorlesung gesehen?

- ☐ Die Heimatverzeichnisse stehen immer in `/usr`.
- ☐ Das primäre Promptsymbol ist weiterhin durch `$` vorbesetzt.
- ☐ Die verschlüsselten Passwörter stehen in `/etc/passwd`.
- ☐ Der Super-user hat weiterhin den login-Namen `root`.

Aufgabe 2: Welche der folgenden Aussagen zur Passwort-Datei `/etc/passwd` sind richtig?

- ☐ Nur der Super-User kann den Inhalt der Datei lesen.
- ☐ Jeder Benutzer kann nur seinen eigenen Eintrag in der Passwort-Datei auflisten, die Einträge der anderen Benutzer bleiben aus Sicherheitsgründen verborgen.
- ☐ Jeder Benutzer des Systems kann den Inhalt der Datei lesen.
- ☐ Jeder Benutzer kann sein Passwort durch Editieren der Datei ändern (Alternative zum Kommando `passwd`).
- ☐ In der Datei wird jedem Benutzer ein Heimatverzeichnis zugeordnet.
- ☐ Die Passwort-Datei legt fest, welchen Gruppen die einzelnen Benutzer angehören.
- ☐ Durch das Hinzufügen eines Punkts am Ende eines Eintrags kann festgelegt werden, dass das jeweilige Arbeitsverzeichnis in den Suchpfad des Benutzers einbezogen wird.

Aufgabe 3: Welche Aussagen zu Hard und Soft Links sind richtig?

- ☐ Hard Links können auch auf Verzeichnisse zeigen.
- ☐ Die Anzahl der Hard Links, die auf eine Datei verweisen, kann man mit Hilfe des Kommandos `ls -l` ermitteln.
- ☐ Beim Verschieben einer Datei, auf die ein Soft Link verweist, muss dieser manuell angepasst werden.
- ☐ Soft Links können nur innerhalb eines Dateisystems eingesetzt werden.

Aufgabe 4: Welche Behauptungen zu i-Knoten (i-nodes) sind richtig?

- ☐ Die Nummern sind nur innerhalb eines Dateisystems gültig.
- ☐ Auch Verzeichnisse haben einen i-Knoten.
- ☐ Nach dem Löschen einer Datei wird die Nummer nie wieder verwendet.
- ☐ Die Nummer kann man sich z. B. mit `ls -i` anzeigen lassen.
- ☐ Dateinamen, die per Hard Link auf die selbe Datei zeigen, zeigen auf den selben i-Knoten.
- ☐ Jedem laufenden Prozess ist ein i-Knoten zugeordnet.

Aufgabe 5: Welche Aussagen sind richtig? Wird in der Shell interaktiv ein Kommando aufgerufen, für das ein Unterprozess erzeugt wird, dann

- ☐ überlagert sich die Shell zuerst mit dem Kommandocode (exec) und verzweigt sich dann (fork).
- ☐ erbt der Unterprozess die Prozessumgebung der Shell einschließlich aller von der Shell erkannten Umlenkungen der Standardeingabe und -ausgabe.
- ☐ analysiert die Shell zunächst die Optionen des Kommandos und erzeugt nur dann den Unterprozess, wenn diese Optionen gültig sind.
- ☐ erhält die Shell anschließend einen Beendigungsstatus 0 (exit status 0) des Kommandos zurück, wenn dieses fehlerfrei beendet wurde.

Aufgabe 6: Was leistet das `export`-Kommando?

- ☐ Die so markierten Variablen mit den gesetzten Werten sind auch in Unterprozessen bekannt.
- ☐ Veränderungen in Unterprozessen werden an den Vaterprozess zurückgegeben.
- ☐ Die so markierten Variablen werden in `.profile` (oder einer ähnlichen Profile-Datei) im Heimatverzeichnis abgelegt.
- ☐ Erlaubt die Auslagerung von Dateibäumen auf externe Datenträger.

Aufgabe 7: Im Verzeichnis aushang stehen drei Dateien, davon eine mit `ln` erzeugte. Ergänzen Sie die fehlende Ausgabe:

```
wegner@holle:~/kurs/aushang> ls -i
42382 DatenOrganisation  42400 bs  42382 do
wegner@holle:~/kurs/aushang> ls -i | sort

_____________________________________

_____________________________________

_____________________________________
```

Aufgabe 8: Wie kann man für die Datei `myScript` der Gruppe und allen anderen die selben Rechte wie dem Besitzer einräumen.

- ☐ `chmod 777 myScript`
- ☐ `chmod go=u mySkript`
- ☐ `chmod -g=u -o=u myScript`
- ☐ `chown $owner myScript`

Aufgabe 9: Betrachten Sie das folgende Skript `datum`:

```
set `date`
shift 1
echo $# ist immer ein Argument
```

Was liefert der Aufruf unten?

```
wegner@holle:~> date
Mi Jul 13 12:04:03 CEST 2005
wegner@holle:~>datum heute
_____________________________________________________
wegner@holle:~>
```

Aufgabe 10: Soll bei einem Shell-Skript z. B. die Standardausgabe unterdrückt werden, lenkt man sie in `/dev/null` um. Kann man für die **Standardeingabe** auch `/dev/null` angeben?

- ☐ Nein, akzeptiert die Shell nicht.
- ☐ Ja, liefert sofort Dateiende.
- ☐ Nur wenn `/dev/null` auf die Tastatur umgelenkt wurde.
- ☐ Nur wenn man vorher etwas in `/dev/null` abgespeichert hat.

Aufgabe 11: Im aktuellen Verzeichnis befinde sich nur die Datei `myFile`. Diese soll in `yourFile` umbenannt werden. Welche der folgenden Kommandozeilen können dazu verwendet werden?

- ☐ `ln myFile yourFile; rm myFile`
- ☐ `rm myFile; cp myFile yourFile`
- ☐ `ren myFile yourFile`
- ☐ `mv myFile yourFile`

Aufgabe 12: Die LINUX-Version hat erfreulicherweise die Option `-x` für den Editor `vi` zum Verschlüsseln einer Datei. Welche der folgenden Aussagen sind richtig.

- ☐ Beim erstmaligen Anlegen wird die zweimalige Eingabe eines Schlüssels verlangt.
- ☐ Die verschlüsselte Datei kann auch nicht mehr ausgegeben oder gelesen werden (wie ein `-r` für ugo).
- ☐ Der Super-user kann weiterhin den Klartext lesen ohne Kenntnis des Schlüssels.
- ☐ Anders als beim Passwort, kann man den Super-user **nicht** bitten, den Schlüssel auf einen neuen Wert zu setzten, wenn man den Schlüssel vergessen hat.

Aufgabe 13: Zentraler Bestandteil des UNIX-Sicherheitskonzepts ist das so genannte Set-User-Id-Bit (kurz: s-Bit). Welche der folgenden Behauptungen sind korrekt?

- ☐ Nur der Super-User kann das s-Bit für eine Datei setzen.
- ☐ Beim Passwortkommando `/bin/passwd` ist das s-Bit gesetzt.
- ☐ Mit dem Kommando `chmod u+s myFile` wird das s-Bit für die Datei `myFile` gesetzt.
- ☐ Ein s-Bit an einer Shell (z. B. `bash`) mit Besitzer `root` wäre absolut harmlos, weil alle Shells gut getestete Programme sind.
- ☐ Kommandos mit gesetztem s-Bit dürfen nur vom Super-User ausgeführt werden.

Aufgabe 14: In der Shell seien die Variablen a und b wie folgt definiert:

```
a="echo hallo"
b=wc
```

Betrachten Sie den folgenden Ablauf. Wie lautet die fehlende Ausgabe?

```
wegner@holle:~/bin> echo "$a | $b"
echo hallo | wc
wegner@holle:~/bin> $a | $b
      1       1       6
wegner@holle:~/bin> echo $a | $b

_____________________________________________________

wegner@holle:~/bin>
```

Aufgabe 15: Der Preis für das abartigste Shell-Skript des Jahres geht in 2005 an den folgenden Zweizeiler `wunder`. Die Zeilen unten zeigen das wundersame Verhalten, das mit jedem Aufruf mehr Ausgabezeilen produziert.

```
wegner@holle:~/bin> wunder geschehen immer wieder
geschehen immer wieder
Oh Wunder
wegner@holle:~/bin> wunder geschehen immer wieder
geschehen immer wieder
Oh Wunder
Oh Wunder
wegner@holle:~/bin> wunder geschehen immer wieder
geschehen immer wieder
Oh Wunder
Oh Wunder
Oh Wunder
wegner@holle:~/bin>
```

Ergänzen Sie im ursprünglichen wunderbaren Skript die fehlende Stelle!

```
# Shell-Skript wunder
echo $*
echo 'echo Oh Wunder' ____________ $0
```

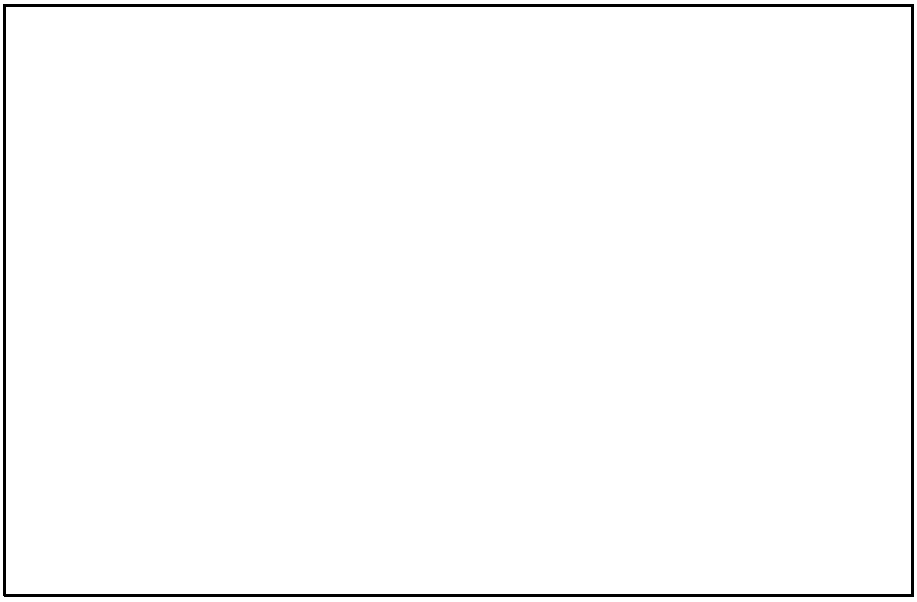
Aufgabe 16: Welche Ausgabe liefert das Shell-Skript `punkte` unten, wenn es mit der Zeile

```
fix@holle:~> punkte nu mach mal nen .
```

aufgerufen wird.

```
# Shell-Skript punkte
echo $1
punkte=""
while [ $# -ne 1 ]
do
  shift 1
  echo $punkte"."$1
  punkte=$punkte"."
done
```

Ausgabe



Prof. Dr. Lutz Wegner
Dipl.-Ing. Nabil Benamar 15. Juli 2005

Einführung in UNIX

Klausur zum Sommersemester 2005

MUSTERLÖSUNG

Name: ..... Vorname: .....

Matr. Nr.:..... Fachbereich:.....

- Bearbeiten Sie alle Fragen!
- Bei Ankreuzaufgaben kann mehr als eine Antwort richtig sein.
- Hilfsmittel sind nicht zugelassen!
- Falls Sie für die Beantwortung der Aufgaben zusätzlichen Platz benötigen, verwenden Sie die Rückseiten. Separate Blätter sind nicht zugelassen.

Aufgabe	Mögliche Punkte	Erreichte Punkte
1	2	
2	3	
3	4	
4	4	
5	4	
6	2	
7	3	
8	3	
9	3	
10	2	
11	4	
12	4	
13	4	
14	4	
15	4	
16	4	
Summe	54	

Aufgabe 1: Manche Dinge werden mit dem Alter immer besser. Das gilt auch für UNIX. Was gilt in allen Versionen und Varianten (einschließlich LINUX), wie in der Vorlesung gesehen?

- ☐ Die Heimatverzeichnisse stehen immer in `/usr`.
- ☐ Das primäre Promptsymbol ist weiterhin durch `$` vorbesetzt.
- ☐ Die verschlüsselten Passwörter stehen in `/etc/passwd`.
- ☒ Der Super-user hat weiterhin den login-Namen `root`.

Aufgabe 2: Welche der folgenden Aussagen zur Passwort-Datei `/etc/passwd` sind richtig?

- ☐ Nur der Super-User kann den Inhalt der Datei lesen.
- ☐ Jeder Benutzer kann nur seinen eigenen Eintrag in der Passwort-Datei auflisten, die Einträge der anderen Benutzer bleiben aus Sicherheitsgründen verborgen.
- ☒ Jeder Benutzer des Systems kann den Inhalt der Datei lesen.
- ☐ Jeder Benutzer kann sein Passwort durch Editieren der Datei ändern (Alternative zum Kommando `passwd`).
- ☒ In der Datei wird jedem Benutzer ein Heimatverzeichnis zugeordnet.
- ☐ Die Passwort-Datei legt fest, welchen Gruppen die einzelnen Benutzer angehören.
- ☐ Durch das Hinzufügen eines Punkts am Ende eines Eintrags kann festgelegt werden, dass das jeweilige Arbeitsverzeichnis in den Suchpfad des Benutzers einbezogen wird.

Aufgabe 3: Welche Aussagen zu Hard und Soft Links sind richtig?

- ☐ Hard Links können auch auf Verzeichnisse zeigen.
- ☒ Die Anzahl der Hard Links, die auf eine Datei verweisen, kann man mit Hilfe des Kommandos `ls -l` ermitteln.
- ☒ Beim Verschieben einer Datei, auf die ein Soft Link verweist, muss dieser manuell angepasst werden.
- ☐ Soft Links können nur innerhalb eines Dateisystems eingesetzt werden.

Aufgabe 4: Welche Behauptungen zu i-Knoten (i-nodes) sind richtig?

- ☒ Die Nummern sind nur innerhalb eines Dateisystems gültig.
- ☒ Auch Verzeichnisse haben einen i-Knoten.
- ☐ Nach dem Löschen einer Datei wird die Nummer nie wieder verwendet.
- ☒ Die Nummer kann man sich z. B. mit `ls -i` anzeigen lassen.
- ☒ Dateinamen, die per Hard Link auf die selbe Datei zeigen, zeigen auf den selben i-Knoten.
- ☐ Jedem laufenden Prozess ist ein i-Knoten zugeordnet.

Aufgabe 5: Welche Aussagen sind richtig? Wird in der Shell interaktiv ein Kommando aufgerufen, für das ein Unterprozess erzeugt wird, dann

- ☐ überlagert sich die Shell zuerst mit dem Kommandocode (exec) und verzweigt sich dann (fork).
- ☒ erbt der Unterprozess die Prozessumgebung der Shell einschließlich aller von der Shell erkannten Umlenkungen der Standardeingabe und -ausgabe.
- ☐ analysiert die Shell zunächst die Optionen des Kommandos und erzeugt nur dann den Unterprozess, wenn diese Optionen gültig sind.
- ☒ erhält die Shell anschließend einen Beendigungsstatus 0 (exit status 0) des Kommandos zurück, wenn dieses fehlerfrei beendet wurde.

Aufgabe 6: Was leistet das `export`-Kommando?

- ☒ Die so markierten Variablen mit den gesetzten Werten sind auch in Unterprozessen bekannt.
- ☐ Veränderungen in Unterprozessen werden an den Vaterprozess zurückgegeben.
- ☐ Die so markierten Variablen werden in `.profile` (oder einer ähnlichen Profile-Datei) im Heimatverzeichnis abgelegt.
- ☐ Erlaubt die Auslagerung von Dateibäumen auf externe Datenträger.

Aufgabe 7: Im Verzeichnis aushang stehen drei Dateien, davon eine mit `ln` erzeugte. Ergänzen Sie die fehlende Ausgabe:

```
wegner@holle:~/kurs/aushang> ls -i
42382 DatenOrganisation  42400 bs  42382 do
wegner@holle:~/kurs/aushang> ls -i | sort
```

42382 DatenOrganisation

42382 do

42400 bs

Aufgabe 8: Wie kann man für die Datei `myScript` der Gruppe und allen anderen die selben Rechte wie dem Besitzer einräumen.

- ☐ `chmod 777 myScript`
- ☒ `chmod go=u myScript`
- ☐ `chmod -g=u -o=u myScript`
- ☐ `chown $owner myScript`

Aufgabe 9: Betrachten Sie das folgende Skript `datum`:

```
set `date`
shift 1
echo $# ist immer ein Argument
```

Was liefert der Aufruf unten?

```
wegner@holle:~> date
Mi Jul 13 12:04:03 CEST 2005
wegner@holle:~>datum heute
```

5 ist immer ein Argument

```
wegner@holle:~>
```

Aufgabe 10: Soll bei einem Shell-Skript z. B. die Standardausgabe unterdrückt werden, lenkt man sie in `/dev/null` um. Kann man für die **Standardeingabe** auch `/dev/null` angeben?

- ☐ Nein, akzeptiert die Shell nicht.
- ☒ Ja, liefert sofort Dateiende.
- ☐ Nur wenn `/dev/null` auf die Tastatur umgelenkt wurde.
- ☐ Nur wenn man vorher etwas in `/dev/null` abgespeichert hat.

Aufgabe 11: Im aktuellen Verzeichnis befinde sich nur die Datei `myFile`. Diese soll in `yourFile` umbenannt werden. Welche der folgenden Kommandozeilen können dazu verwendet werden?

- ☒ `ln myFile yourFile; rm myFile`
- ☐ `rm myFile; cp myFile yourFile`
- ☐ `ren myFile yourFile`
- ☒ `mv myFile yourFile`

Aufgabe 12: Die LINUX-Version hat erfreulicherweise die Option `-x` für den Editor `vi` zum Verschlüsseln einer Datei. Welche der folgenden Aussagen sind richtig.

- ☒ Beim erstmaligen Anlegen wird die zweimalige Eingabe eines Schlüssels verlangt.
- ☐ Die verschlüsselte Datei kann auch nicht mehr ausgegeben oder gelesen werden (wie ein `-r` für ugo).
- ☐ Der Super-user kann weiterhin den Klartext lesen ohne Kenntnis des Schlüssels.
- ☒ Anders als beim Passwort, kann man den Super-user **nicht** bitten, den Schlüssel auf einen neuen Wert zu setzten, wenn man den Schlüssel vergessen hat.

Aufgabe 13: Zentraler Bestandteil des UNIX-Sicherheitskonzepts ist das so genannte Set-User-Id-Bit (kurz: s-Bit). Welche der folgenden Behauptungen sind korrekt?

- ☐ Nur der Super-User kann das s-Bit für eine Datei setzen.
- ☒ Beim Passwortkommando `/bin/passwd` ist das s-Bit gesetzt.
- ☒ Mit dem Kommando `chmod u+s myFile` wird das s-Bit für die Datei `myFile` gesetzt.
- ☐ Ein s-Bit an einer Shell (z. B. `bash`) mit Besitzer `root` wäre absolut harmlos, weil alle Shells gut getestete Programme sind.
- ☐ Kommandos mit gesetztem s-Bit dürfen nur vom Super-User ausgeführt werden.

Aufgabe 14: In der Shell seien die Variablen a und b wie folgt definiert:

```
a="echo hallo"
b=wc
```

Betrachten Sie den folgenden Ablauf. Wie lautet die fehlende Ausgabe?

```
wegner@holle:~/bin> echo "$a | $b"
echo hallo | wc
wegner@holle:~/bin> $a | $b
      1       1       6
wegner@holle:~/bin> echo $a | $b


     1          2          11
wegner@holle:~/bin>
```

Aufgabe 15: Der Preis für das abartigste Shell-Skript des Jahres geht in 2005 an den folgenden Zweizeiler `wunder`. Die Zeilen unten zeigen das wundersame Verhalten, das mit jedem Aufruf mehr Ausgabezeilen produziert.

```
wegner@holle:~/bin> wunder geschehen immer wieder
geschehen immer wieder
Oh Wunder
wegner@holle:~/bin> wunder geschehen immer wieder
geschehen immer wieder
Oh Wunder
Oh Wunder
wegner@holle:~/bin> wunder geschehen immer wieder
geschehen immer wieder
Oh Wunder
Oh Wunder
Oh Wunder
wegner@holle:~/bin>
```

Ergänzen Sie im ursprünglichen wunderbaren Skript die fehlende Stelle!

```
# Shell-Skript wunder
echo $*
echo 'echo Oh Wunder' ____>>_______ $0
```

Aufgabe 16: Welche Ausgabe liefert das Shell-Skript `punkte` unten, wenn es mit der Zeile

```
fix@holle:~> punkte nu mach mal nen .
```

aufgerufen wird.

```
# Shell-Skript punkte
echo $1
punkte=""
while [ $# -ne 1 ]
do
  shift 1
  echo $punkte"."$1
  punkte=$punkte"."
done
```

Ausgabe

```
nu
.mach
..mal
...nen
.....
```

Prof. Dr. Lutz Wegner
Dipl.-Ing. Nabil Benamar
Dipl.-Math Dipl. Inf. Sebastian Pape **28. Februar 2006**

Einführung in UNIX
Klausur zum Wintersemester 2005/06

Name:.....Vorname:.....

Matr. Nr.:.....Studiengang:.....

Bearbeiten Sie alle Fragen! Bei Ankreuzaufgaben kann mehr als eine Antwort richtig sein. Hilfsmittel sind nicht zugelassen! Falls Sie für die Beantwortung der Aufgaben zusätzlichen Platz benötigen, verwenden Sie die Rückseiten. Separate Blätter sind nicht zugelassen.

Aufgabe	Punkte max.	Punkte erreicht
1	2	
2	2	
3	2	
4	2	
5	2	
6	2	
7	2	
8	2	
9	2	
10	2	
11	4	
12	2	
13	4	
14	2	
15	4	
16	2	
Summe	**38**	

Aufgabe 1

Unix gibt es in kommerziellen Varianten, z.B. Solaris und AIX, sowie in frei verfügbaren, z.B. Linux und FreeBSD. Muß man je nach Variante eine bestimmte Shell - z.B. eine von `csh`, `ksh`, `sh`, `bash` - verwenden?

( ) Nein.

( ) Ja immer.

( ) Ja, aber nur bei den kommerziellen Varianten.

( ) Ja, aber nur bei den frei verfügbaren Varianten.

Aufgabe 2

Der vermutlich gefährlichste und dümmste Dateiname wäre „`-r *`“, so man ihn erzeugen könnte. Kann man ihn erzeugen?

( ) Ja, leider, z.B. mit `echo Aber hallo >'-r *'`.

( ) Nein, geht nicht wegen Leerzeichen im Namen.

( ) Nein, geht nicht wegen Minuszeichen im Namen.

( ) Nein, geht nicht wegen Stern im Namen.

( ) Nein, geht aus mehreren Gründen nicht.

Aufgabe 3

Das `kill`-Kommando sendet ein Signal an einen Prozeß, der dadurch zum Abbruch gebracht wird, sofern das Signal nicht von diesem Prozeß abgefangen werden kann. Dabei gilt:

( ) Es wird ein Zombie-Prozeß erzeugt.

( ) `init` adoptiert den abgebrochenen Prozeß.

( ) Man kann nur eigene Prozesse abbrechen.

( ) Man kann nur interaktive Vordergrundprozesse abbrechen.

( ) Man muß den Prozeßidentifier des abzubrechenden Prozesses kennen.

Aufgabe 4

Welche Aussagen sind zu harten und weichen Links (Dateiverweisen) richtig?

( ) Harte Links kann man nur innerhalb eines Dateisystems (Volume) setzen.

( ) Weiche Links darf man nicht innerhalb eines Dateisystems (Volume) setzen.

( ) Die Link-Zählung, die man z.B. bei `ls -l` sieht, berücksichtigt nur harte Links.

( ) Eine Datei, auf die kein weicher Link mehr zeigt, wird automatisch gelöscht.

Aufgabe 5

Erzeugt die Shell einen Unterprozeß, z.B. für ein Kommando, werden für diesen Sohnprozeß die Belegungen für die Dateideskriptoren vorab gelöscht, damit der Sohnprozeß diese je nach Anwendung neu setzen kann.

( ) Nein, werden vererbt vom Vater.

( ) Nein, außer Sohnprozeß ist Hintergrundprozeß.

( ) Ja, aber nur die Deskriptoren 0, 1 und 2, die übrigen bleiben bestehen.

( ) Ja, alle außer 0, 1 und 2.

( ) Hängt von der Shell ab.

Aufgabe 6

Was wird dem login-Namen zugeordnet, wenn der Systemadministrator einen neuen Teilnehmer anlegt?

( ) ein User-Id.

( ) eine Login-Gruppe.

( ) ein Heimatverzeichnis.

( ) eine effektive UID für Aufrufe von Kommandos mit SUID-Bit.

( ) ein speziell auf ihn angepasstes Promptsymbol

Aufgabe 7

Kann ein unvorsichtiger Benutzer in einem Shell-Skript eine vorbesetzte Systemvariable analog zu `HOME` oder `PATH` unbewußt verwenden und neu setzen?

( ) Ja, könnte gefährlich oder überraschend für die Ausführung des Skripts sein.

( ) Ja, ist aber immer harmlos, solange diese nicht exportiert wird.

( ) Nein, Anwender und Systemvariablen sind getrennte Namensräume.

( ) Nein, wird bei der Ausführung entdeckt; das Skript bricht ab und liefert Exitstatus 1.

Aufgabe 8

Wäre `read` kein Spezialkommando der Shell (built-in command), hätte man nach dem Aufruf der Kommandozeile `read eingabe` und der Rückkehr zur Shell keinen Wert in der Variablen `eingabe`. Richtig?

( ) Ja, liegt an der Einbahnstraße für die Werteweitergabe.

( ) Nein, könnte man mit `export` umgehen.

( ) Nein, könnte man mit `import` umgehen.

( ) Das Problem stellt sich nicht, weil `read` nur in Shellskripten auftreten darf.

( ) Das Problem stellt sich nicht, weil `read` als Kommando nicht existiert, sondern nur als Systemaufruf (system call).

Aufgabe 9

Für eine Datei sei z.B. der Besitzer `fix` und die Besitzergruppe `stkom` (Studienkommission) eingetragen. Die Dateirechte für `u`, `g`, `o` lassen sich dann von wem neu setzen?

( ) Nur vom Besitzer (`fix`) alle Rechte.

( ) Vom Besitzer alle Rechte, von jedem in der Gruppe (`stkom`) nur die Gruppenrechte.

( ) Von jedem in der Gruppe (`stkom`) alle Rechte unabhängig von der gegenwärtigen Gruppenkennung.

( ) Von jedem in der Gruppe (`stkom`) alle Rechte, wenn vorher die Gruppenkennung mit `newgrp` angenommen wurde.

( ) Hängt vom SGID (set group-owner ID) Bit ab.

( ) Hängt davon ab, ob es eine Normaldatei, Verzeichnis oder Gerät ist.

Aufgabe 10

Idealerweise legt Unix für die Hintereinanderschaltung von Kommandos in einer Pipe nur kleine Zwischenpuffer an, weil die Daten wie am Fließband durch die Kommandos laufen. Bei welchen Kommandos in einer Pipeline wird man wohl große Puffer (oder von Unix angelegte temporäre Dateien) brauchen, wenn große Datenströme fließen?

( ) `... | sort | ...`

( ) `... | uniq | ...`

( ) `... | head -10000c | ...`

( ) `... | tail -100000c | ...`

( ) `... | wc | ...`

Aufgabe 11

Die Datei `schneewittchen` enthalte die folgenden Daten:

```
Spieglein, Spieglein
an der Wand,
wer ist der Klügste
im ganzen Land?
```

Das Shellskript `meinekatze` habe den folgenden Inhalt, wobei `echo -e` die Interpretation von `\n` als newline (NL, neue Zeile) übernimmt..

```
while true
do
    read eingabe
    if [ "$eingabe" = "" ]
    then break
    fi
    ausgabe=$eingabe'\n'$ausgabe
done
echo -ne $ausgabe
```

Wie lautet die Ausgabe beim Aufruf von `meinekatze <schneewittchen` ?

Aufgabe 12

Greifen Sie nach den Sternen! Mit `touch xyz` kann man das Modifikationsdatum der Datei `xyz` auf das gegenwärtige Datum (und die gegenwärtige Uhrzeit) setzen. Wie könnte man das Datum aller Einträge im gegenwärtigen Verzeichnis mit `touch` setzen?

Aufgabe 13

Analog zu Aufgabe 12 ist ein kleines, aber gefährliches Shell-Skript `leermacher` gesucht, das mittels `echo -n >xyz` alle Dateien `xyz` in einem Verzeichnis leermacht. Der Aufruf von `leermacher` soll mit genau einem Argumente erfolgen, das ein Verzeichnis ist. Ergänzen Sie die fehlenden Stellen!

```
if [ $# -ne 1 ]
then echo leermacher: nur ein Argument
  exit 1
fi
if [ ____________________ ]
then
    cd ______________________________
    set `ls -a`
    for i
    do
        if [ ____________________ ]
        then echo ____________________
        fi
    done
else
    echo leermacher: Argument muss Verzeichnis sein
    exit 1
fi
```

Aufgabe 14

Ein bekanntes Betriebssystem verschiebt beim Löschen von Dateien und Verzeichnissen diese in den Papierkorb, den man dann gesondert „leeren“ muß. Schreiben Sie ein Shellskript `myremove`, das beim Aufruf mit einem oder mehreren Argumenten diese in ein Verzeichnis `.Papierkorb` verschiebt, das im Heimatverzeichnis liegt.

Aufgabe 15

Modifizieren Sie `myremove` aus Aufgabe 14, damit es bei Aufruf mit der Option `-f` („force“) nicht verschiebt, sondern wirklich löscht.

Aufgabe 16

Mit welchen der folgenden Kommandos bekommt man den gegenwärtigen Katalog selbst (nicht seinen Inhalt) in der langen Form mit `ls` angezeigt?

( ) `ls -ld`

( ) `ls -dl`

( ) `ls -d-l` *Hinweis: kein Blank zwischen den Optionen!*

( ) `ls -l -d`

( ) `ls -ll .`

( ) `ls -dd .`

ENDE DER KLAUSUR

Prof. Dr. Lutz Wegner
Dipl.-Ing. Nabil Benamar
Dipl.-Math Dipl. Inf. Sebastian Pape **28. Februar 2006**

Einführung in UNIX
Klausur zum Wintersemester 2005/06

MUSTERLÖSUNG

Name:.....Vorname:.....

Matr. Nr.:.....Studiengang:.....

Bearbeiten Sie alle Fragen! Bei Ankreuzaufgaben kann mehr als eine Antwort richtig sein. Hilfsmittel sind nicht zugelassen! Falls Sie für die Beantwortung der Aufgaben zusätzlichen Platz benötigen, verwenden Sie die Rückseiten. Separate Blätter sind nicht zugelassen.

Aufgabe	Punkte max.	Punkte erreicht
1	2	
2	2	
3	2	
4	2	
5	2	
6	2	
7	2	
8	2	
9	2	
10	2	
11	4	
12	2	
13	4	
14	2	
15	4	
16	2	
Summe	**38**	

Aufgabe 1

Unix gibt es in kommerziellen Varianten, z.B. Solaris und AIX, sowie in frei verfügbaren, z.B. Linux und FreeBSD. Muß man je nach Variante eine bestimmte Shell - z.B. eine von `csh`, `ksh`, `sh`, `bash` - verwenden?

(X) Nein.

( ) Ja immer.

( ) Ja, aber nur bei den kommerziellen Varianten.

( ) Ja, aber nur bei den frei verfügbaren Varianten.

Aufgabe 2

Der vermutlich gefährlichste und dümmste Dateiname wäre „`-r *`“, so man ihn erzeugen könnte. Kann man ihn erzeugen?

(X) Ja, leider, z.B. mit `echo Aber hallo >'-r *'`.

( ) Nein, geht nicht wegen Leerzeichen im Namen.

( ) Nein, geht nicht wegen Minuszeichen im Namen.

( ) Nein, geht nicht wegen Stern im Namen.

( ) Nein, geht aus mehreren Gründen nicht.

Aufgabe 3

Das `kill`-Kommando sendet ein Signal an einen Prozeß, der dadurch zum Abbruch gebracht wird, sofern das Signal nicht von diesem Prozeß abgefangen werden kann. Dabei gilt:

( ) Es wird ein Zombie-Prozeß erzeugt.

( ) `init` adoptiert den abgebrochenen Prozeß.

(X) Man kann nur eigene Prozesse abbrechen.

( ) Man kann nur interaktive Vordergrundprozesse abbrechen.

(X) Man muß den Prozeßidentifier des abzubrechenden Prozesses kennen.

Aufgabe 4

Welche Aussagen sind zu harten und weichen Links (Dateiverweisen) richtig?

(X) Harte Links kann man nur innerhalb eines Dateisystems (Volume) setzen.

( ) Weiche Links darf man nicht innerhalb eines Dateisystems (Volume) setzen.

(X) Die Link-Zählung, die man z.B. bei `ls -l` sieht, berücksichtigt nur harte Links.

( ) Eine Datei, auf die kein weicher Link mehr zeigt, wird automatisch gelöscht.

Aufgabe 5

Erzeugt die Shell einen Unterprozeß, z.B. für ein Kommando, werden für diesen Sohnprozeß die Belegungen für die Dateideskriptoren vorab gelöscht, damit der Sohnprozeß diese je nach Anwendung neu setzen kann.

(X) Nein, werden vererbt vom Vater.

( ) Nein, außer Sohnprozeß ist Hintergrundprozeß.

( ) Ja, aber nur die Deskriptoren 0, 1 und 2, die übrigen bleiben bestehen.

( ) Ja, alle außer 0, 1 und 2.

( ) Hängt von der Shell ab.

Aufgabe 6

Was wird dem login-Namen zugeordnet, wenn der Systemadministrator einen neuen Teilnehmer anlegt?

(X) ein User-Id.

(X) eine Login-Gruppe.

(X) ein Heimatverzeichnis.

( ) eine effektive UID für Aufrufe von Kommandos mit SUID-Bit.

( ) ein speziell auf ihn angepasstes Promptsymbol

Aufgabe 7

Kann ein unvorsichtiger Benutzer in einem Shell-Skript eine vorbesetzte Systemvariable analog zu `HOME` oder `PATH` unbewußt verwenden und neu setzen?

(X) Ja, könnte gefährlich oder überraschend für die Ausführung des Skripts sein.

( ) Ja, ist aber immer harmlos, solange diese nicht exportiert wird.

( ) Nein, Anwender und Systemvariablen sind getrennte Namensräume.

( ) Nein, wird bei der Ausführung entdeckt; das Skript bricht ab und liefert Exitstatus 1.

Aufgabe 8

Wäre `read` kein Spezialkommando der Shell (built-in command), hätte man nach dem Aufruf der Kommandozeile `read eingabe` und der Rückkehr zur Shell keinen Wert in der Variablen `eingabe`. Richtig?

(X) Ja, liegt an der Einbahnstraße für die Werteweitergabe.

( ) Nein, könnte man mit `export` umgehen.

( ) Nein, könnte man mit `import` umgehen.

( ) Das Problem stellt sich nicht, weil `read` nur in Shellskripten auftreten darf.

( ) Das Problem stellt sich nicht, weil `read` als Kommando nicht existiert, sondern nur als Systemaufruf (system call).

Aufgabe 9

Für eine Datei sei z.B. der Besitzer `fix` und die Besitzergruppe `stkom` (Studienkommission) eingetragen. Die Dateirechte für `u`, `g`, `o` lassen sich dann von wem neu setzen?

(X) Nur vom Besitzer (`fix`) alle Rechte.

( ) Vom Besitzer alle Rechte, von jedem in der Gruppe (`stkom`) nur die Gruppenrechte.

( ) Von jedem in der Gruppe (`stkom`) alle Rechte unabhängig von der gegenwärtigen Gruppenkennung.

( ) Von jedem in der Gruppe (`stkom`) alle Rechte, wenn vorher die Gruppenkennung mit `newgrp` angenommen wurde.

( ) Hängt vom SGID (set group-owner ID) Bit ab.

( ) Hängt davon ab, ob es eine Normaldatei, Verzeichnis oder Gerät ist.

Aufgabe 10

Idealerweise legt Unix für die Hintereinanderschaltung von Kommandos in einer Pipe nur kleine Zwischenpuffer an, weil die Daten wie am Fließband durch die Kommandos laufen. Bei welchen Kommandos in einer Pipeline wird man wohl große Puffer (oder von Unix angelegte temporäre Dateien) brauchen, wenn große Datenströme fließen?

(X) `... | sort | ...`

( ) `... | uniq | ...`

( ) `... | head -10000c | ...`

(X) `... | tail -100000c | ...`

( ) `... | wc | ...`

Aufgabe 11

Die Datei `schneewittchen` enthalte die folgenden Daten:

```
Spieglein, Spieglein
an der Wand,
wer ist der Klügste
im ganzen Land?
```

Das Shellskript `meinekatze` habe den folgenden Inhalt, wobei `echo -e` die Interpretation von `\n` als newline (NL, neue Zeile) übernimmt..

```
while true
do
    read eingabe
    if [ "$eingabe" = "" ]
    then break
    fi
    ausgabe=$eingabe'\n'$ausgabe
done
echo -ne $ausgabe
```

Ausgabe

```
im ganzen Land?
wer ist der Klügste
an der Wand,
Spieglein, Spieglein
```

Wie lautet die Ausgabe beim Aufruf von `meinekatze <schneewittchen` ?

Aufgabe 12

Greifen Sie nach den Sternen! Mit `touch xyz` kann man das Modifikationsdatum der Datei `xyz` auf das gegenwärtige Datum (und die gegenwärtige Uhrzeit) setzen. Wie könnte man das Datum aller Einträge im gegenwärtigen Verzeichnis mit `touch` setzen?

```
touch .* *
```

Aufgabe 13

Analog zu Aufgabe 12 ist ein kleines, aber gefährliches Shell-Skript `leermacher` gesucht, das mittels `echo -n >xyz` alle Dateien `xyz` in einem Verzeichnis leermacht. Der Aufruf von `leermacher` soll mit genau einem Argumente erfolgen, das ein Verzeichnis ist. Ergänzen Sie die fehlenden Stellen!

```
if [ $# -ne 1 ]
then echo leermacher: nur ein Argument
  exit 1
fi
if [ -d $1 ]
then
    cd $1
    set `ls -a`
    for i
    do
        if [ -f $i ]
        then echo -n >$i
        fi
    done
else
    echo leermacher: Argument muss Verzeichnis sein
    exit 1
fi
```

Aufgabe 14

Ein bekanntes Betriebssystem verschiebt beim Löschen von Dateien und Verzeichnissen diese in den Papierkorb, den man dann gesondert „leeren“ muß. Schreiben Sie ein Shellskript `myremove`, das beim Aufruf mit einem oder mehreren Argumenten diese in ein Verzeichnis `.Papierkorb` verschiebt, das im Heimatverzeichnis liegt.

```
mv $* $HOME/.Papierkorb
```

Aufgabe 15

Modifizieren Sie `myremove` aus Aufgabe 14, damit es bei Aufruf mit der Option `-f` („force“) nicht verschiebt, sondern wirklich löscht.

```
if [ "$1" = "-f"]
then shift 1
     rm -r $*
else mv $* $HOME/.Papierkorb
fi
```

Aufgabe 16

Mit welchen der folgenden Kommandos bekommt man den gegenwärtigen Katalog selbst (nicht seinen Inhalt) in der langen Form mit `ls` angezeigt?

(X) `ls -ld`

(X) `ls -dl`

( ) `ls -d-l` *Hinweis: kein Blank zwischen den Optionen!*

(X) `ls -l -d`

( ) `ls -ll .`

( ) `ls -dd .`

ENDE DER KLAUSUR

Prof. Dr. Lutz Wegner
Dipl.-Ing. Nabil Benamar
Dipl.-Math Dipl. Inf. Sebastian Pape **28. Juli 2006**

Einführung in UNIX
Klausur zum Sommersemester 2006

Name:.....Vorname:.....

Matr. Nr.:.....Studiengang:.....

Bearbeiten Sie alle Fragen! Bei Ankreuzaufgaben kann mehr als eine Antwort richtig sein. Hilfsmittel sind nicht zugelassen! Falls Sie für die Beantwortung der Aufgaben zusätzlichen Platz benötigen, verwenden Sie die Rückseiten. Separate Blätter sind nicht zugelassen.

Aufgabe	Punkte max.	Punkte erreicht
1	2	
2	2	
3	2	
4	2	
5	2	
6	2	
7	2	
8	6	
9	6	
10	4	
11	3	
12	5	
13	2	
Summe	**40**	

Aufgabe 1

Heutige Unix oder Linux Systeme haben einen sog. History-Mechanismus, bei dem man mit der Pfeiltaste (Pfeil nach oben) vorherige Kommandozeilen am Prompt einfügen kann.

( ) Ja, wird auf Shellebene erledigt.

( ) Ja, schließt auch ein „rückgängig machen“ (undo, Shift+Pfeiltaste) ein, begrenzt auf die Länge der History-Liste.

( ) Ja, wird auf Kernel-Ebene erledigt, da auch die alten Argumentersetzungen gespeichert und bei nochmaligem Aufruf eingesetzt werden.

( ) Ja, setzt aber eine graphische Oberfläche (Desktop und Papierkorb) voraus.

Aufgabe 2

In unserer Sammlung gefährlicher und dummer Dateinamen betrachten wir heute den Namen, der nur aus einem Anführungszeichen (") besteht. Kann man ihn erzeugen?

( ) Ja, kann man z.B. mit `touch \"` erzeugen.

( ) Ja, kann man z.B. mit `touch '"'` erzeugen.

( ) Ja, kann man z.B. mit `touch "` erzeugen.

( ) Ja, kann man z.B. mit `touch ""` erzeugen.

( ) Nein, geht nicht.

Aufgabe 3

Ein Zombie-Prozeß

( ) kann man mit `kill -9` nicht beseitigen.

( ) wird immer von `init` adoptiert.

( ) wartet darauf, daß der Vater mit `wait()` seinen Exit-Status abruft.

( ) hat schon seine Segmente (text, heap, stack) abgegeben.

( ) hat keinen Besitzer mehr und gehört deshalb `root`.

Aufgabe 4

Welche Aussagen sind zu harten Links (Dateiverweisen) richtig?

( ) Harte Links kann man nur innerhalb eines Dateisystems (Volume) setzen.

( ) Ein harter Link darf nicht auf ein Verzeichnis gesetzt werden.

( ) Die Link-Zählung, die man z.B. bei `ls -l` sieht, berücksichtigt nur harte Links.

( ) Den Dateinamen, unter dem eine Datei angelegt wurde, kann man löschen, ohne daß der Datei verschwindet, wenn ein zweiter Name (Alias) als harter Link auf sie zeigt.

Aufgabe 5

Entwirft man unter Unix ein Programm, dann muß man sich vorher festlegen, ob man die Standardausgabe in eine Datei, das Terminal oder eine Pipe schreibt.

( ) Nein, man schreibt immer in den Deskriptor 1 (stdout).

( ) Nein, aber es kann dann passieren, daß das Schreiben nicht funktioniert.

( ) Ja, Deskriptor 1 für Pipe, 2 für Terminal, >2 für Datei.

( ) Ja, wenn die Ausgabe ASCII ist..

( ) Hängt von der Shell ab.

Aufgabe 6

Im Kurs setzen wir oft die Dateirechte oktal, z.B. mit `chmod 644`. Tatsächlich sind es eigentlich 4 Oktalziffern, die man angeben kann, meist läßt man aber das *set user ID bit*, das *set group ID* und *sticky bit* weg. Man könnte sogar `chmod 44 seltsam` oder `chmod 4 nochseltsamer` angeben. Demnach gilt wohl:

( ) Fehlende Ziffern werden links mit 0 aufgefüllt.

( ) Fehlende Ziffern werden rechts mit 0 aufgefüllt.

( ) Fehlende Ziffern werden links mit 7 aufgefüllt.

( ) Fehlende Ziffern werden rechts mit 7 aufgefüllt.

Aufgabe 7

In einem Verzeichnis seien einige XML Dokumente, allerdings fehle den Dateinamen überall das Suffix „`.xml`“. Dieses kann man allen Dateien anfügen wie folgt, wenn man schon im Verzeichnis steht:

( ) `mv * *.xml`

( ) `for n in \`ls .\`; do mv $n $n.xml; done`

( ) `mv \* \*.xml . ;# nach Verzeichnis Punkt (.)`

( ) `for n in echo *; do mv $n $n.xml; done`

Aufgabe 8

Wir haben Shell-Skript `notenvergabe` geschrieben, das ein Bild beurteilt und dafür eine Note zwischen 1 und 6 vergibt, die auf die Standardausgabe geschrieben wird. Mit `notenvergabe` *bildname* wird es aufgerufen.

Betrachten Sie jetzt das Shell-Skript `myviewer` unten, das genauso wie `notenvergabe` im Suchpfad liegt. Im Arbeitsverzeichnis befinden sich die drei Dateien

```
anne.jpg beate.jpg colette.jpg
```

Was liefert der Aufruf `myviewer`, wenn `notenvergabe` allen Bildern eine 2 gibt?

```
dateien=`ls .`
mkdir top gehtso wegdamit
for dat in $dateien
do
  if [ -f $dat ]
  then
    note=`notenvergabe $dat`
    if [ $note -le "2" ]
      then mv $dat top
      elif [ $note -le "4" ]
        then mv $dat gehtso
        else mv $dat wegdamit
    fi
  fi
done
```

Aufgabe 9

Das folgende Shellskript `create_index` verlangt genau ein Argument beim Aufruf. Unter der Annahme, wir seien im Verzeichnis `UrlaubMalle1999`, das genau drei Dateien

```
anne.jpg beate.jpg colette.jpg
```

enthält, was bewirkt der Aufruf „`create_index bilder`“ ?

```
i=1
for datei in `ls .`
do
  echo "$i $datei" >>$1.index
  i=`expr $i + 1`
done
```

Aufgabe 10

In der `bash` existiert eine Variable `RANDOM`, die eine Zufallszahl zwischen 0 und 32767 liefert. Ein Shellskript `spieler` bestehe nur aus der Zeile `echo $RANDOM`.

Schreiben Sie ein Shellskript `bank`, das aus der Standardeingabe eine Zahl liest und mit einer selbst erzeugten Zufallszahl vergleicht. Ist die gelesene Zahl größer, erfolgt die Ausgabe „Spieler gewinnt“, sonst die Ausgabe „Bank gewinnt“. Der Aufruf erfolgt mit `spieler | bank`.

Aufgabe 11:

Das Skript `lstree` soll den Inhalt eines Verzeichnisses rekursiv in die Datei `/tmp/lstreeOutput` ausgeben. Leider steht in der Datei immer nur eine Zeile.

Ändern Sie das Skript so, dass es richtig arbeitet.

lstree

```
cd $1
for i in * do
    if test -d $i
        then
            echo $tab\| $i >/tmp/lstreeOutput
            tab="--"$tab lstree $i
        elif test -f $i
            then
                echo $tab\| $i >/tmp/lstreeOutput
    fi
done
```

Aufgabe 12:

In dem aktuellen Arbeitsverzeichnis befinden sich nur die Dateien "1", "2" und "3" mit den Rechten 644, die dem eingeloggten Benutzer gehören. Welche der folgenden Zeilen gibt ein Fragezeichen aus, welche einen Stern. Was wird in den anderen Fällen ausgegeben?

```
echo "?"; rm *        _____________________________
echo ?; rm *          _____________________________
rm *; echo ?          _____________________________
rm *; echo $?         _____________________________
rm *; echo *          _____________________________
```

Aufgabe 13

In dem aktuellen Arbeitsverzeichnis befinden sich nur die Dateien "1","2" und "3". Der eingeloggte Benutzer hat alle Rechte, um die Dateien zu löschen. Welcher der folgenden Befehle löscht alle Dateien?

```
(  ) for i in *;do rm $i;done
(  ) (rm *)
(  ) [rm *]
(  ) rm `echo *`
(  ) echo * > rm
```

ENDE DER KLAUSUR

Prof. Dr. Lutz Wegner
Dipl.-Ing. Nabil Benamar
Dipl.-Math Dipl. Inf. Sebastian Pape **28. Juli 2006**

Einführung in UNIX
Klausur zum Sommersemester 2006

MUSTERLÖSUNG

Name:.....Vorname:.....

Matr. Nr.:.....Studiengang:.....

Bearbeiten Sie alle Fragen! Bei Ankreuzaufgaben kann mehr als eine Antwort richtig sein. Hilfsmittel sind nicht zugelassen! Falls Sie für die Beantwortung der Aufgaben zusätzlichen Platz benötigen, verwenden Sie die Rückseiten. Separate Blätter sind nicht zugelassen.

Aufgabe	Punkte max.	Punkte erreicht
1	2	
2	2	
3	2	
4	2	
5	2	
6	2	
7	2	
8	6	
9	6	
10	4	
11	3	
12	5	
13	2	
Summe	**40**	

Aufgabe 1

Heutige Unix oder Linux Systeme haben einen sog. History-Mechanismus, bei dem man mit der Pfeiltaste (Pfeil nach oben) vorherige Kommandozeilen am Prompt einfügen kann.

(X) Ja, wird auf Shellebene erledigt.

( ) Ja, schließt auch ein „rückgängig machen“ (undo, Shift+Pfeiltaste) ein, begrenzt auf die Länge der History-Liste.

( ) Ja, wird auf Kernel-Ebene erledigt, da auch die alten Argumentersetzungen gespeichert und bei nochmaligem Aufruf eingesetzt werden.

( ) Ja, setzt aber eine graphische Oberfläche (Desktop und Papierkorb) voraus.

Aufgabe 2

In unserer Sammlung gefährlicher und dummer Dateinamen betrachten wir heute den Namen, der nur aus einem Anführungszeichen (") besteht. Kann man ihn erzeugen?

(X) Ja, kann man z.B. mit `touch \"` erzeugen.

(X) Ja, kann man z.B. mit `touch '"'` erzeugen.

( ) Ja, kann man z.B. mit `touch "` erzeugen.

( ) Ja, kann man z.B. mit `touch ""` erzeugen.

( ) Nein, geht nicht.

Aufgabe 3

Ein Zombie-Prozeß

(X) kann man mit `kill -9` nicht beseitigen.

( ) wird immer von `init` adoptiert.

(X) wartet darauf, daß der Vater mit `wait()` seinen Exit-Status abruft.

(X) hat schon seine Segmente (text, heap, stack) abgegeben.

( ) hat keinen Besitzer mehr und gehört deshalb `root`.

Aufgabe 4

Welche Aussagen sind zu harten Links (Dateiverweisen) richtig?

(X) Harte Links kann man nur innerhalb eines Dateisystems (Volume) setzen.

(X) Ein harter Link darf nicht auf ein Verzeichnis gesetzt werden.

(X) Die Link-Zählung, die man z.B. bei `ls -l` sieht, berücksichtigt nur harte Links.

(X) Den Dateinamen, unter dem eine Datei angelegt wurde, kann man löschen, ohne daß der Datei verschwindet, wenn ein zweiter Name (Alias) als harter Link auf sie zeigt.

Aufgabe 5

Entwirft man unter Unix ein Programm, dann muß man sich vorher festlegen, ob man die Standardausgabe in eine Datei, das Terminal oder eine Pipe schreibt.

(X) Nein, man schreibt immer in den Deskriptor 1 (stdout).

( ) Nein, aber es kann dann passieren, daß das Schreiben nicht funktioniert.

( ) Ja, Deskriptor 1 für Pipe, 2 für Terminal, >2 für Datei.

( ) Ja, wenn die Ausgabe ASCII ist..

( ) Hängt von der Shell ab.

Aufgabe 6

Im Kurs setzen wir oft die Dateirechte oktal, z.B. mit `chmod 644`. Tatsächlich sind es eigentlich 4 Oktalziffern, die man angeben kann, meist läßt man aber das *set user ID bit*, das *set group ID* und *sticky bit* weg. Man könnte sogar `chmod 44 seltsam` oder `chmod 4 nochseltsamer` angeben. Demnach gilt wohl:

(X) Fehlende Ziffern werden links mit 0 aufgefüllt.

( ) Fehlende Ziffern werden rechts mit 0 aufgefüllt.

( ) Fehlende Ziffern werden links mit 7 aufgefüllt.

( ) Fehlende Ziffern werden rechts mit 7 aufgefüllt.

Aufgabe 7

In einem Verzeichnis seien einige XML Dokumente, allerdings fehle den Dateinamen überall das Suffix „`.xml`“. Dieses kann man allen Dateien anfügen wie folgt, wenn man schon im Verzeichnis steht:

( ) `mv * *.xml`

(X) `for n in \`ls .\`; do mv $n $n.xml; done`

( ) `mv \* \*.xml . ;# nach Verzeichnis Punkt (.)`

( ) `for n in echo *; do mv $n $n.xml; done`

bei Kreuz hier kein Abzug

Aufgabe 8

Wir haben Shell-Skript `notenvergabe` geschrieben, das ein Bild beurteilt und dafür eine Note zwischen 1 und 6 vergibt, die auf die Standardausgabe geschrieben wird. Mit `notenvergabe` *bildname* wird es aufgerufen.

Betrachten Sie jetzt das Shell-Skript `myviewer` unten, das genauso wie `notenvergabe` im Suchpfad liegt. Im Arbeitsverzeichnis befinden sich die drei Dateien

```
anne.jpg beate.jpg colette.jpg
```

Was liefert der Aufruf `myviewer`, wenn `notenvergabe` allen Bildern eine 2 gibt?

```
dateien=`ls .`
mkdir top gehtso wegdamit
for dat in $dateien
do
  if [ -f $dat ]
  then
    note=`notenvergabe $dat`
    if [ $note -le "2" ]
      then mv $dat top
      elif [ $note -le "4" ]
        then mv $dat gehtso
        else mv $dat wegdamit
    fi
  fi
done
```

Erzeugt die drei Verzeichnisse top, gehtso und wegdamit im Arbeitsverzeichnis. Im Beispiel hier werden die drei Bilder nach top verschoben.

Aufgabe 9

Das folgende Shellskript `create_index` verlangt genau ein Argument beim Aufruf. Unter der Annahme, wir seien im Verzeichnis `UrlaubMalle1999`, das genau drei Dateien

```
anne.jpg beate.jpg colette.jpg
```

enthält, was bewirkt der Aufruf „`create_index bilder`“ ?

```
i=1
for datei in `ls .`
do
  echo "$i $datei" >>$1.index
  i=`expr $i + 1`
done
```

Erzeugt eine Datei bilder.index, in der drei Zeilen stehen. Jede Zeile enthält einen Indexwert (1, 2, ...) gefolgt von einem Dateinamen aus dem Arbeitsverzeichnis. Hier also

1 anne.jpg

2 beate.jpg

3 colette.jpg

Aufgabe 10

In der `bash` existiert eine Variable `RANDOM`, die eine Zufallszahl zwischen 0 und 32767 liefert. Ein Shellskript `spieler` bestehe nur aus der Zeile `echo $RANDOM`.

Schreiben Sie ein Shellskript `bank`, das aus der Standardeingabe eine Zahl liest und mit einer selbst erzeugten Zufallszahl vergleicht. Ist die gelesene Zahl größer, erfolgt die Ausgabe „Spieler gewinnt“, sonst die Ausgabe „Bank gewinnt“. Der Aufruf erfolgt mit `spieler | bank`.

```
read ein
r=$RANDOM
if [ $ein -gt $r ]
    then echo Spieler gewinnt
    else echo Bank gewinnt
fi
```

Aufgabe 11:

Das Skript `lstree` soll den Inhalt eines Verzeichnisses rekursiv in die Datei `/tmp/lstreeOutput` ausgeben. Leider steht in der Datei immer nur eine Zeile.

Ändern Sie das Skript so, dass es richtig arbeitet.

lstree

```
cd $1
for i in * do
    if test -d $i
        then
            echo $tab\| $i >>/tmp/lstreeOutput
            tab="--"$tab lstree $i
        elif test -f $i
            then
                echo $tab\| $i >>/tmp/lstreeOutput
    fi
done
```

Aufgabe 12:

In dem aktuellen Arbeitsverzeichnis befinden sich nur die Dateien "1", "2" und "3" mit den Rechten 644, die dem eingeloggten Benutzer gehören. Welche der folgenden Zeilen gibt ein Fragezeichen aus, welche einen Stern. Was wird in den anderen Fällen ausgegeben?

`echo "?"; rm *`	**?**
`echo ?; rm *`	**1 2 3**
`rm *; echo ?`	**?**
`rm *; echo $?`	**0**
`rm *; echo *`	*****

Aufgabe 13

In dem aktuellen Arbeitsverzeichnis befinden sich nur die Dateien "1","2" und "3". Der eingeloggte Benutzer hat alle Rechte, um die Dateien zu löschen. Welcher der folgenden Befehle löscht alle Dateien?

(X) `for i in *;do rm $i;done`

(X) `(rm *)`

( ) `[rm *]`

(X) `rm \`echo *\``

( ) `echo * > rm`

ENDE DER KLAUSUR

Prof. Dr. Lutz Wegner
Dipl.-Math. Dipl.-Inf. Sebastian Pape **27. Oktober 2006**

Einführung in UNIX
Klausur zum Wintersemester 2006/07

Name:.....Vorname:.....

Matr. Nr.:.....Studiengang:.....

Bearbeiten Sie alle Fragen! Bei Ankreuzaufgaben kann mehr als eine Antwort richtig sein. Hilfsmittel sind nicht zugelassen! Falls Sie für die Beantwortung der Aufgaben zusätzlichen Platz benötigen, verwenden Sie die Rückseiten. Separate Blätter sind nicht zugelassen.

Aufgabe	Punkte max.	Punkte erreicht
1	2	
2	2	
3	2	
4	2	
5	2	
6	2	
7	2	
8	3	
9	2	
10	2	
11	3	
12	2	
13	2	
14	2	
15	2	
16	3	
17	4	
18	3	
19	3	
Summe	**45**	

Aufgabe 1

Welchen Zweck erfüllt eine `.profile` (oder ähnlich benannte) Datei im Heimatverzeichnis?

( ) Erlaubt das Zurückholen vorher eingegebener Kommandozeilen mit den Cursor-Tasten (history).

( ) Vorinitialisierung von Umgebungsvariablen und andere Aufgaben zu Sitzungsbeginn.

( ) Prüft das Passwort des Benutzers und stellt die Gruppenberechtigung her.

( ) Dateien wie `.profile` oder ähnlich benannte zum Speichern von Benutzereinstellungen von Sitzung zu Sitzung gibt es heute aus Sicherheitsgründen nicht mehr.

Aufgabe 2

Wo wir gerade bei der versteckten Normaldatei `.profile` aus dem Kurs sind: Kann es eigentlich auch versteckte Verzeichnisse geben?

( ) Ja, z.B. „`.`“ und „`..`“ .

( ) Ja, kann man sich mit `ls -a` anzeigen lassen.

( ) Nein, geht prinzipiell nicht.

( ) Kann man nicht mit ja oder nein beantworten, hängt von der Shell ab.

Aufgabe 3

In unserer Sammlung gefährlicher und dummer Dateinamen betrachten wir heute den Namen, der nur aus drei Punkten (`...`) besteht. Kann man ihn erzeugen?

( ) Ja, kann man z.B. mit `touch \.\.\.` erzeugen.

( ) Ja, kann man z.B. mit `touch "..."` erzeugen.

( ) Ja, kann man sogar ganz locker mit `touch ...` erzeugen.

( ) Nein, ist alias von `etc` und damit ein Verzeichnis.

( ) Nein, geht nicht wegen `"."` und `".."`.

Aufgabe 4

Ein Prozeß wird zum „Waisenkind“ wenn

( ) der ursprüngliche Vaterprozeß vor ihm gestorben ist.

( ) `init` stirbt.

( ) der Vater nicht mit `wait()` seinen Exit-Status abruft.

( ) der Prozeß seine Segmente (text, heap, stack) abgegeben hat.

( ) keinen Besitzer mehr hat und deshalb `root` gehört.

Aufgabe 5

Die Kommandoeingabe `(date; ps)` erzeugt die folgende Ausgabe:

```
Mit Sep 27 12:11:11 CEST 2006
  PID TTY          TIME CMD
29738 pts/0    00:00:00 bash
29869 pts/0    00:00:00 bash
29871 pts/0    00:00:00 ps
wegner@haensel:~>
```

Welche der folgenden Aussagen ist/sind richtig?

( ) Die Ausführung von `ps` erfolgt wegen der Klammern in einer Untershell.

( ) Die Ausführung von `date` erfolgt wegen der Klammern in einer Untershell.

( ) Als interaktive Shell und als Untershell dient die `bash`.

( ) Die Ausgabe von `ps` zeigt auch den Prozeß für `ps` selbst.

( ) Da bei der Ausführung offensichtlich keine Fehler auftraten, würde ein nachfolgendes `echo $?` (*exit status* des letzten ausgeführten Kommandos) den Wert 0 liefern.

Aufgabe 6

Gehört ein Benutzer mehreren Gruppen an, dann möchte er manchmal beim Arbeiten am Rechner seine Gruppenidentität wechseln. Welche Aussage ist richtig?

( ) Dazu muß man sich neu anmelden.

( ) Geht mit `newgrp`.

( ) Geht grundsätzlich nicht, weil es nur Gruppenrechte von Dateien und keine Gruppenidentität von Prozessen gibt.

( ) Geht nur für *root*.

( ) Gibt es nicht mehr, weil der Wechsel eine Sicherheitslücke darstellt.

Aufgabe 7

Das `date`-Kommando liefert mit der Option `-r` *Dateiname* die Modifikationszeit der Datei. Man betrachte die folgende Kommandozeile (mit Lücke) und die beiden Ausgabezeilen.

```
~/aushang$ date; ____________________; date -r test
Sa Okt 14 17:28:13 CEST 2006
Sa Okt 14 17:28:13 CEST 2006
~/aushang$
```

Ergänzen Sie die Lücke so, daß ein möglicher Inhalt der Datei `test` nicht verändert wird! Hinweis: es gibt mehrere mögliche Antworten, eine richtige, möglichst kurze genügt.

Aufgabe 8

Harte und weiche Links (Dateiverweise) zu kombinieren ist sicherlich nicht sehr schlau. Aus dem Ablauf unten kann man aber sehen, wie Linux damit umgeht. Welche der nachfolgenden Aussagen sind richtig?

```
wegner@haensel:~/kurs/links> echo hallo >frage
wegner@haensel:~/kurs/links> ln frage hart
wegner@haensel:~/kurs/links> ln -s hart weich
wegner@haensel:~/kurs/links> ls -l
insgesamt 8
-rw-r--r--    2 wegner   mitarb        6 Sep 29 10:53 frage
-rw-r--r--    2 wegner   mitarb        6 Sep 29 10:53 hart
lrwxrwxrwx    1 wegner   mitarb        4 Sep 29 10:55 weich -> hart
wegner@haensel:~/kurs/links> ln weich ganzhart
ln: »weich«: Warnung: Erstellen einer harten Verknüpfung auf eine
symbolische Verknüpfung ist nicht portabel
wegner@haensel:~/kurs/links> ls -l
insgesamt 8
-rw-r--r--    2 wegner   mitarb         6 Sep 29 10:53 frage
lrwxrwxrwx    2 wegner   mitarb         4 Sep 29 10:55 ganzhart ->
hart
-rw-r--r--    2 wegner   mitarb         6 Sep 29 10:53 hart
lrwxrwxrwx    2 wegner   mitarb         4 Sep 29 10:55 weich ->
hart
wegner@haensel:~/kurs/links> ls -i
1196088 frage  1196092 ganzhart  1196088 hart  1196092 weich
```

( ) Das Setzen des weichen Verweises `weich` auf `hart` erhöht nicht den Verweiszähler von `frage` und `hart`.

( ) Wegen der Warnung wird der Verweis `ganzhart` nicht gesetzt.

( ) Durch den Verweis `ganzhart` erhöht sich nochmals der Linkzähler für `frage` und `hart`.

( ) Der Verweis `ganzhart` wird als harter Link auf `weich` gesetzt.

( ) Obwohl `ganzhart` als harter Verweis auf `weich` gesetzt werden soll, legt Linux ihn als einen weichen Verweis auf `hart` an.

Aufgabe 9

Der Rio500 war einer der ersten MP3 Player mit USB-Anschluß, der 1999 auf den Markt kam. Unter Linux ist der Spieler ein Gerät, den man (z.B. auf unserem Rechner `rapunzel`) unter welchem Dateieintrag anspricht?

( ) `/dev/usb/rio500`

( ) `/usr/etc/rio500`

( ) `/bin/rio500` (mit Option `-usb`)

( ) `/audio/mp3`

( ) ->Systemsteuerung ->Audio- und Soundeinstellung->Neuen Treiber laden

Aufgabe 10

Seit der „Erfindung“ von Unix in den Sechzigern und heute haben sich die Erwartungen an ein leistungsfähiges Dateisystem geändert. Beispiele sind lange Dateinamen (z.B. max. 255 Zeichen statt 14), große Dateien (z.B. max. 2 GB statt 64 MB), große Dateisysteme (z.B. max. 4 TB statt 64 MB). Welche Änderungen am Dateisystem waren hierzu nötig?

( ) Das Konzept der i-Knoten (inodes) als Bindeglied zwischen Verzeichnissen und physischem Dateisystem mußte aufgegeben werden.

( ) Das Konzept der Verzeichnisse als Dateien mit einer Auflistung der Dateinamen und i-Nummern mußte aufgegeben werden.

( ) Die bei `ls -i` sichtbaren i-Nummern (inumbers) wurden intern von 16 auf 32 bit Integer umgestellt.

( ) Die interne Struktur der i-Knoten (inodes) wurde angepaßt, z.B. für Dateigrößen und Blockadressen.

( ) Alle Kommandos, die mit dem Dateisystem zu tun haben (z.B. `cp`, `mv`, `ln`, `ls`, ...) wurden grundsätzlich geändert.

( ) Das Konzept der harten Verweise (hard links) mußte aufgegeben werden.

Aufgabe 11

Welche Ausgabe liefert das folgende Shell-Skript. Achtung: $1$3 nicht $1$2 !

```
set Gesundheits Reform Kopf Pauschale Bürger Versicherung
while [ $# -gt 2 ]
do
    echo $1$3
    shift
done
```

Aufgabe 12

Die Ausgabe von `ls | wc` laute

```
16       16       121
```

Wie lautet die Ausgabe von echo `‘ls‘ | wc` in der selben Situation?

_____

Aufgabe 13

Ein im Suchpfad für Kommandos befindliches, ausführbares Shell-Skript `seltsam` enthält nur eine Zeile:

```
  echo heller wahnsinn >meintest
```

Aufgerufen mit `seltsam >komisch` ergibt welches Resultat?

( ) nur eine Datei `komisch` mit Inhalt `heller wahnsinn`

( ) nur eine Datei `meintest` mit Inhalt `heller wahnsinn`

( ) eine leere Datei `meintest` und eine Datei `komisch` mit Inhalt `heller wahnsinn`

( ) eine leere Datei `komisch` und eine Datei `meintest` mit Inhalt `heller wahnsinn`

( ) zwei leere Dateien `komisch` und `meintest`

Aufgabe 14

Was ist richtig: Ein *here-Dokument* in einem Shellskript `myscript`

( ) ersetzt Argumente aus dem Aufruf von `myscript` durch die Wörter des Dokuments

( ) ersetzt in der Kommandozeile mit dem Aufruf `myscript` die Kommandozeile mit der Ausgabe des *here-Dokuments* unter Austausch von newline durch blank.

( ) macht das *here-Dokument* zur Standardeingabe von `myscript`.

( ) erzeugt in `myscript` ein Kommando, das sich aus der Auswertung (`eval`) des *here-Dokuments* ergibt.

Aufgabe 15

Wie im Kurs gesehen, war die Verwendung von `crypt` nicht ganz einfach, weil das dahintersteckende Kommando `mcrypt`, bzw. `mdecrypt` verschiedene Optionen hat, unter anderem auch für den verwendeten Algorithmus, z.B. DES oder enigma. Grundsätzlich richtig ist aber:

( ) Verschlüsselung und Entschlüsselung müssen über den selben Algorithmus laufen.

( ) Mehrfache Verschlüsselung ist möglich.

( ) Die Datei mit dem Klartext wird man nach dem Verschlüsseln in der Regel löschen.

( ) Das `crypt`-Kommando ist verträglich mit der `x`-Option einiger Editoren.

( ) Auch verschlüsselte Datein können vom super-user (root) als Klartext gelesen werden.

Aufgabe 16

Die `man`-Seiten zu `touch` zeigen, daß man das Modifikations- und Lesedatum auch auf einen beliebigen anderen Wert (vor- oder zurück-)setzen kann.

```
TOUCH(1)            User Commands                TOUCH(1)

NAME
    touch - change file timestamps

SYNOPSIS
    touch [OPTION]... FILE...

DESCRIPTION
    Update the access and modification times of each
    FILE to the current time.
...

-t STAMP
    use [[CC]YY]MMDDhhmm[.ss] instead of current time
```

Setzen Sie das Datum der Datei `meintouch` auf den Tag und die Startuhrzeit dieser Klausur.

_____

Aufgabe 17

Wir wollen interaktiv ein Datum im Format der vorherigen Aufgabe einlesen. In `/tmp` soll dann eine Datei `timestamp` angelegt werden, deren Modifikationszeit mit `touch -t ...` auf diese Zeit gesetzt wird. Die so erzeugte Datei mit dem Zeitstempel können wir für Sicherungszwecke verwenden. Vervollständigen Sie das Shell-Skript `setzestempel` unten. Auf eine Prüfung der Eingaben verzichten wir zunächst.

```
stempel=""

echo -n Bitte Jahr in der Form YYYY eingeben" ":

read wert; _____

echo -n _____:

_____

_____

_____

_____

_____

_____

_____

touch -t _____
```

Aufgabe 18

In dem Shellskript `setzestempel` oben liefert `touch -t` *stamp file* einen Fehler und legt die Datei *file* nicht an, bzw. ändert das Datum einer bestehenden Datei nicht, wenn die Zeitangabe *stamp* ein falsches Format hat. Geben Sie unten die neue letzte Zeile in `setzestempel` an, sodaß nach `touch -t` ... nur dann die Zeile

```
Datei /tmp/timestamp angelegt
```

auf der Standardausgabe ausgegeben wird, wenn `touch` erfolgreich war.

Aufgabe 19

Vervollständigen Sie das folgende Shellskript `backup`, das als erstes Argument ein Verzeichnis und als zweites Argument eine Referenzdatei bekommt. Das Shell-Skript durchläuft das Verzeichnis und gibt alle Dateieinträge darin auf der Standardausgabe aus, die jüngeren Datums als die Referenzdatei sind. Der Einfachheit halber kann angenommen werden, daß keine versteckten Dateien und keine Unterverzeichnisse im übergebenen Verzeichnis enthalten sind.

Man verwende dazu den bedingten Ausdruck *file1* `-nt` *file2* , der wahr liefert, wenn *file1* jünger (nach dem Modifikationsdatum) ist als *file2* oder wenn *file1* existiert und *file2* nicht (was hier nicht interessiert).

```
for eintrag in ____________________________

do

if [ ___________________________________ ]

then ___________________________________

fi

done
```

ENDE DER KLAUSUR

Prof. Dr. Lutz Wegner
Dipl.-Math. Dipl.-Inf. Sebastian Pape **27. Oktober 2006**

Einführung in UNIX
Klausur zum Wintersemester 2006/07

Musterlösung

Name:.....Vorname:.....

Matr. Nr.:.....Studiengang:.....

Bearbeiten Sie alle Fragen! Bei Ankreuzaufgaben kann mehr als eine Antwort richtig sein. Hilfsmittel sind nicht zugelassen! Falls Sie für die Beantwortung der Aufgaben zusätzlichen Platz benötigen, verwenden Sie die Rückseiten. Separate Blätter sind nicht zugelassen.

Aufgabe	Punkte max.	Punkte erreicht
1	2	
2	2	
3	2	
4	2	
5	2	
6	2	
7	2	
8	3	
9	2	
10	2	
11	3	
12	2	
13	2	
14	2	
15	2	
16	3	
17	4	
18	3	
19	3	
Summe	**45**	

Aufgabe 1

Welchen Zweck erfüllt eine `.profile` (oder ähnlich benannte) Datei im Heimatverzeichnis?

( ) Erlaubt das Zurückholen vorher eingegebener Kommandozeilen mit den Cursor-Tasten (history).

(X) Vorinitialisierung von Umgebungsvariablen und andere Aufgaben zu Sitzungsbeginn.

( ) Prüft das Passwort des Benutzers und stellt die Gruppenberechtigung her.

( ) Dateien wie `.profile` oder ähnlich benannte zum Speichern von Benutzereinstellungen von Sitzung zu Sitzung gibt es heute aus Sicherheitsgründen nicht mehr.

Aufgabe 2

Wo wir gerade bei der versteckten Normaldatei `.profile` aus dem Kurs sind: Kann es eigentlich auch versteckte Verzeichnisse geben?

(X) Ja, z.B. „`.`“ und „`..`“ .

(X) Ja, kann man sich mit `ls -a` anzeigen lassen.

( ) Nein, geht prinzipiell nicht.

( ) Kann man nicht mit ja oder nein beantworten, hängt von der Shell ab.

Aufgabe 3

In unserer Sammlung gefährlicher und dummer Dateinamen betrachten wir heute den Namen, der nur aus drei Punkten (`...`) besteht. Kann man ihn erzeugen?

(X) Ja, kann man z.B. mit `touch \.\.\.` erzeugen.

(X) Ja, kann man z.B. mit `touch "..."` erzeugen.

(X) Ja, kann man sogar ganz locker mit `touch ...` erzeugen.

( ) Nein, ist alias von `etc` und damit ein Verzeichnis.

( ) Nein, geht nicht wegen `"."` und `".."`.

Aufgabe 4

Ein Prozeß wird zum „Waisenkind“ wenn

(X) der ursprüngliche Vaterprozeß vor ihm gestorben ist.

( ) `init` stirbt.

( ) der Vater nicht mit `wait()` seinen Exit-Status abruft.

( ) der Prozeß seine Segmente (text, heap, stack) abgegeben hat.

( ) keinen Besitzer mehr hat und deshalb `root` gehört.

Aufgabe 5

Die Kommandoeingabe `(date; ps)` erzeugt die folgende Ausgabe:

```
Mit Sep 27 12:11:11 CEST 2006
  PID TTY          TIME CMD
29738 pts/0    00:00:00 bash
29869 pts/0    00:00:00 bash
29871 pts/0    00:00:00 ps
wegner@haensel:~>
```

Welche der folgenden Aussagen ist/sind richtig?

(X) Die Ausführung von `ps` erfolgt wegen der Klammern in einer Untershell.

(X) Die Ausführung von `date` erfolgt wegen der Klammern in einer Untershell.

(X) Als interaktive Shell und als Untershell dient die `bash`.

(X) Die Ausgabe von `ps` zeigt auch den Prozeß für `ps` selbst.

(X) Da bei der Ausführung offensichtlich keine Fehler auftraten, würde ein nachfolgendes `echo $?` (*exit status* des letzten ausgeführten Kommandos) den Wert 0 liefern.

Aufgabe 6

Gehört ein Benutzer mehreren Gruppen an, dann möchte er manchmal beim Arbeiten am Rechner seine Gruppenidentität wechseln. Welche Aussage ist richtig?

( ) Dazu muß man sich neu anmelden.

(X) Geht mit `newgrp`.

( ) Geht grundsätzlich nicht, weil es nur Gruppenrechte von Dateien und keine Gruppenidentität von Prozessen gibt.

( ) Geht nur für *root*.

( ) Gibt es nicht mehr, weil der Wechsel eine Sicherheitslücke darstellt.

Aufgabe 7

Das `date`-Kommando liefert mit der Option `-r` *Dateiname* die Modifikationszeit der Datei. Man betrachte die folgende Kommandozeile (mit Lücke) und die beiden Ausgabezeilen.

```
~/aushang$ date; touch test ; date -r test
Sa Okt 14 17:28:13 CEST 2006
Sa Okt 14 17:28:13 CEST 2006
~/aushang$
```

Ergänzen Sie die Lücke so, daß ein möglicher Inhalt der Datei `test` nicht verändert wird! Hinweis: es gibt mehrere mögliche Antworten, eine richtige, möglichst kurze genügt.

Aufgabe 8

Harte und weiche Links (Dateiverweise) zu kombinieren ist sicherlich nicht sehr schlau. Aus dem Ablauf unten kann man aber sehen, wie Linux damit umgeht. Welche der nachfolgenden Aussagen sind richtig?

```
wegner@haensel:~/kurs/links> echo hallo >frage
wegner@haensel:~/kurs/links> ln frage hart
wegner@haensel:~/kurs/links> ln -s hart weich
wegner@haensel:~/kurs/links> ls -l
insgesamt 8
-rw-r--r--    2 wegner   mitarb        6 Sep 29 10:53 frage
-rw-r--r--    2 wegner   mitarb        6 Sep 29 10:53 hart
lrwxrwxrwx    1 wegner   mitarb        4 Sep 29 10:55 weich -> hart
wegner@haensel:~/kurs/links> ln weich ganzhart
ln: »weich«: Warnung: Erstellen einer harten Verknüpfung auf eine
symbolische Verknüpfung ist nicht portabel
wegner@haensel:~/kurs/links> ls -l
insgesamt 8
-rw-r--r--    2 wegner   mitarb         6 Sep 29 10:53 frage
lrwxrwxrwx    2 wegner   mitarb         4 Sep 29 10:55 ganzhart ->
hart
-rw-r--r--    2 wegner   mitarb         6 Sep 29 10:53 hart
lrwxrwxrwx    2 wegner   mitarb         4 Sep 29 10:55 weich ->
hart
wegner@haensel:~/kurs/links> ls -i
1196088 frage  1196092 ganzhart  1196088 hart  1196092 weich
```

(X) Das Setzen des weichen Verweises `weich` auf `hart` erhöht nicht den Verweiszähler von `frage` und `hart`.

( ) Wegen der Warnung wird der Verweis `ganzhart` nicht gesetzt.

( ) Durch den Verweis `ganzhart` erhöht sich nochmals der Linkzähler für `frage` und `hart`.

(X) Der Verweis `ganzhart` wird als harter Link auf `weich` gesetzt.

( ) Obwohl `ganzhart` als harter Verweis auf `weich` gesetzt werden soll, legt Linux ihn als einen weichen Verweis auf `hart` an.

Aufgabe 9

Der Rio500 war einer der ersten MP3 Player mit USB-Anschluß, der 1999 auf den Markt kam. Unter Linux ist der Spieler ein Gerät, den man (z.B. auf unserem Rechner `rapunzel`) unter welchem Dateieintrag anspricht?

(X) `/dev/usb/rio500`

( ) `/usr/etc/rio500`

( ) `/bin/rio500` (mit Option `-usb`)

( ) `/audio/mp3`

( ) ->Systemsteuerung ->Audio- und Soundeinstellung->Neuen Treiber laden

Aufgabe 10

Seit der „Erfindung“ von Unix in den Sechzigern und heute haben sich die Erwartungen an ein leistungsfähiges Dateisystem geändert. Beispiele sind lange Dateinamen (z.B. max. 255 Zeichen statt 14), große Dateien (z.B. max. 2 GB statt 64 MB), große Dateisysteme (z.B. max. 4 TB statt 64 MB). Welche Änderungen am Dateisystem waren hierzu nötig?

( ) Das Konzept der i-Knoten (inodes) als Bindeglied zwischen Verzeichnissen und physischem Dateisystem mußte aufgegeben werden.

( ) Das Konzept der Verzeichnisse als Dateien mit einer Auflistung der Dateinamen und i-Nummern mußte aufgegeben werden.

(X) Die bei `ls -i` sichtbaren i-Nummern (inumbers) wurden intern von 16 auf 32 bit Integer umgestellt.

(X) Die interne Struktur der i-Knoten (inodes) wurde angepaßt, z.B. für Dateigrößen und Blockadressen.

( ) Alle Kommandos, die mit dem Dateisystem zu tun haben (z.B. `cp`, `mv`, `ln`, `ls`, ...) wurden grundsätzlich geändert.

( ) Das Konzept der harten Verweise (hard links) mußte aufgegeben werden.

Aufgabe 11

Welche Ausgabe liefert das folgende Shell-Skript. Achtung: $1$3 nicht $1$2 !

```
set Gesundheits Reform Kopf Pauschale Bürger Versicherung
while [ $# -gt 2 ]
do
    echo $1$3
    shift
done
```

GesundheitsKopf
ReformPauschale
KopfBürger
PauschaleVersicherung

Aufgabe 12

Die Ausgabe von `ls | wc` laute

```
16       16       121
```

Wie lautet die Ausgabe von echo `'ls' | wc` in der selben Situation?

_____ **1 16 121** _____

Aufgabe 13

Ein im Suchpfad für Kommandos befindliches, ausführbares Shell-Skript `seltsam` enthält nur eine Zeile:

```
  echo heller wahnsinn >meintest
```

Aufgerufen mit `seltsam >komisch` ergibt welches Resultat?

( ) nur eine Datei `komisch` mit Inhalt `heller wahnsinn`

( ) nur eine Datei `meintest` mit Inhalt `heller wahnsinn`

( ) eine leere Datei `meintest` und eine Datei `komisch` mit Inhalt `heller wahnsinn`

(X) eine leere Datei `komisch` und eine Datei `meintest` mit Inhalt `heller wahnsinn`

( ) zwei leere Dateien `komisch` und `meintest`

Aufgabe 14

Was ist richtig: Ein *here-Dokument* in einem Shellskript `myscript`

( ) ersetzt Argumente aus dem Aufruf von `myscript` durch die Wörter des Dokuments

( ) ersetzt in der Kommandozeile mit dem Aufruf `myscript` die Kommandozeile mit der Ausgabe des *here-Dokuments* unter Austausch von newline durch blank.

(X) macht das *here-Dokument* zur Standardeingabe von `myscript`.

( ) erzeugt in `myscript` ein Kommando, das sich aus der Auswertung (`eval`) des *here-Dokuments* ergibt.

Aufgabe 15

Wie im Kurs gesehen, war die Verwendung von `crypt` nicht ganz einfach, weil das dahintersteckende Kommando `mcrypt`, bzw. `mdecrypt` verschiedene Optionen hat, unter anderem auch für den verwendeten Algorithmus, z.B. DES oder enigma. Grundsätzlich richtig ist aber:

(X) Verschlüsselung und Entschlüsselung müssen über den selben Algorithmus laufen.

(X) Mehrfache Verschlüsselung ist möglich.

(X) Die Datei mit dem Klartext wird man nach dem Verschlüsseln in der Regel löschen.

(X) Das `crypt`-Kommando ist verträglich mit der `x`-Option einiger Editoren.

( ) Auch verschlüsselte Datein können vom super-user (root) als Klartext gelesen werden.

Aufgabe 16

Die man-Seiten zu touch zeigen, daß man das Modifikations- und Lesedatum auch auf einen beliebigen anderen Wert (vor- oder zurück-)setzen kann.

```
TOUCH(1)            User Commands                TOUCH(1)

NAME
    touch - change file timestamps

SYNOPSIS
    touch [OPTION]... FILE...

DESCRIPTION
    Update the access and modification times of each
    FILE to the current time.
...

-t STAMP
    use [[CC]YY]MMDDhhmm[.ss] instead of current time
```

Setzen Sie das Datum der Datei meintouch auf den Tag und die Startuhrzeit dieser Klausur.

touch -t 200610271500 meintouch

richtig natürlich auch mit .Sekunden und ohne 20. Jahrhundert

Aufgabe 17

Wir wollen interaktiv ein Datum im Format der vorherigen Aufgabe einlesen. In /tmp soll dann eine Datei timestamp angelegt werden, deren Modifikationszeit mit touch -t ... auf diese Zeit gesetzt wird. Die so erzeugte Datei mit dem Zeitstempel können wir für Sicherungszwecke verwenden. Vervollständigen Sie das Shell-Skript setzestempel unten. Auf eine Prüfung der Eingaben verzichten wir zunächst.

```
stempel=""

echo -n Bitte Jahr in der Form YYYY eingeben" ":

read wert; stempel=$stempel$wert

echo -n Bitte Monat in der Form MM eingeben" ":

read wert; stempel=$stempel$wert

echo -n Bitte Tag in der Form DD eingeben" ":

read wert; stempel=$stempel$wert

...

_____

_____

_____

touch -t $stempel /tmp/timestamp
```

Aufgabe 18

In dem Shellskript `setzestempel` oben liefert `touch -t` *stamp file* einen Fehler und legt die Datei *file* nicht an, bzw. ändert das Datum einer bestehenden Datei nicht, wenn die Zeitangabe *stamp* ein falsches Format hat. Geben Sie unten die neue letzte Zeile in `setzestempel` an, sodaß nach `touch -t` ... nur dann die Zeile

```
Datei /tmp/timestamp angelegt
```

auf der Standardausgabe ausgegeben wird, wenn `touch` erfolgreich war.

touch -t $stempel /tmp/timestamp && echo Datei /tmp/timestamp angelegt

Aufgabe 19

Vervollständigen Sie das folgende Shellskript `backup`, das als erstes Argument ein Verzeichnis und als zweites Argument eine Referenzdatei bekommt. Das Shell-Skript durchläuft das Verzeichnis und gibt alle Dateieinträge darin auf der Standardausgabe aus, die jüngeren Datums als die Referenzdatei sind. Der Einfachheit halber kann angenommen werden, daß keine versteckten Dateien und keine Unterverzeichnisse im übergebenen Verzeichnis enthalten sind.

Man verwende dazu den bedingten Ausdruck *file1* `-nt` *file2* , der wahr liefert, wenn *file1* jünger (nach dem Modifikationsdatum) ist als *file2* oder wenn *file1* existiert und *file2* nicht (was hier nicht interessiert).

```
for eintrag in ____ `ls $1` __
do
if [ $eintrag -nt $2 ]
then echo $eintrag
fi
done
```

ENDE DER KLAUSUR

Prof. Dr. Lutz Wegner
Dipl.-Math. Dipl.-Inf. Sebastian Pape **27. September 2007**

Einführung in UNIX
Klausur zum Sommersemester 2007

Name:.....Vorname:.....

Matr. Nr.:.....Studiengang:.....

Bearbeiten Sie alle Fragen! Bei Ankreuzaufgaben kann mehr als eine Antwort richtig sein. Hilfsmittel sind nicht zugelassen! Falls Sie für die Beantwortung der Aufgaben zusätzlichen Platz benötigen, verwenden Sie die Rückseiten.

Aufgabe	Punkte max.	Punkte erreicht
1	2	
2	2	
3	2	
4	2	
5	2	
6	2	
7	2	
8	2	
9	4	
10	2	
11	4	
12	4	
Summe	**30**	

Aufgabe 1

Als versteckte Dateien sind nur zugelassen `.`, `..` und `.profile`.

( ) Ja.

( ) Nein, jeder Dateiname, der mit einem Punkt beginnt, bezeichnet eine versteckte Datei.

( ) Nein, aber nur der super-user darf heutzutage solche versteckten Dateien anlegen.

( ) Hängt von der UNIX/LINUX Variante (Distribution) ab.

Aufgabe 2

Damit man mit einem Shell-Skript eine Umgebungsvariable mit einer Zeile der Art `export LANG=de_DE.iso885915@euro` setzen kann, ...

( ) ... muß das Skript mit dem `source`-Kommando (`.`) ausgeführt werden.

( ) ... darf das Skript nicht mit dem `source`-Kommando (`.`) ausgeführt werden.

( ) ... muß `LANG` eine built-in Variable der Shell sein.

( ) ... muß für das Skript das set-user-id Bit gesetzt sein.

Aufgabe 3

In unserer Sammlung gefährlicher und dummer Dateinamen betrachten wir heute den Namen, der nur aus dem Fragezeichen besteht. Kann man ihn erzeugen?

( ) Ja, kann man z.B. mit `touch \?` erzeugen.

( ) Ja, kann man z.B. mit `touch "?"` erzeugen.

( ) Ja, kann man sogar ganz locker mit `touch ?` erzeugen, allerdings nur, wenn es nicht schon Dateinamen bestehend aus einem Zeichen gibt.

( ) Ja, kann man sogar ganz locker mit `touch ?` erzeugen, auch dann, wenn es schon Dateinamen bestehend aus einem Zeichen gibt.

( ) Nein, `?` ist ein Shell-Metazeichen (Joker) und damit verboten.

Aufgabe 4

Unabhängig von allen möglichen Optionen für das `ps`-Kommando sieht man nie

( ) `ps` selbst.

( ) den `init` Prozess.

( ) Hintergrundprozesse.

( ) einen Prozess, der schon `wait()` aufgerufen hat.

( ) einen Zombie-Prozess.

( ) schon vollständig abgeschlossene Prozesse, die in der Vergangenheit liefen.

Aufgabe 5

Die Dateirechte einer Datei werden wo gespeichert?

( ) Im Verzeichnis, in dem der Dateiname eingetragen ist.

( ) Im i-Knoten (inode) der Datei.

( ) In den ersten zwei Bytes des Datenbereichs (des 1. Datenblocks) der Datei.

( ) In der `ar`-Tabelle (access rights table) in `/dev`.

( ) Hängt davon ab, wie die Datei erzeugt wurde, z.B. ob die Datei durch einen hard-link erzeugt wurde.

Aufgabe 6

Ein Benutzer legt eine Datei `x` an, z.B. mit `touch x`. Sie habe danach per default die Rechte 644. Dann legt er den Dateinamen `y` als harten Link auf `x` mit `ln x y` an. Zuletzt führt er `chmod 755 x` aus. Welche Aussage ist richtig?

( ) `y` war zunächst auch 644 und ist jetzt 755.

( ) `y` war zunächst 644 und bleibt 644.

( ) `y` hat zunächst die Linkrechte (default 400) und behält diese immer.

( ) Keine generelle Aussage möglich, da abhängig von der `umask`-Einstellung.

( ) Geht generell nicht, da Dateirechte für Dateien mit Linkzähler >1 nicht änderbar sind.

Aufgabe 7

Wie initialisiert der Administrator oder super-user eines UNIX-Rechners die Passwörter neuer Teilnehmer auf sichere Weise, bis Teilnehmer diese dann selbst ändern?

( ) Üblich sind einheitliche Vorbelegungen wie `anfang` oder `start`.

( ) Üblich sind individuelle Vorbelegungen mit zufällig ausgewählten Passwörtern, die den neuen Teilnehmern schriftlich mitgeteilt werden.

( ) Jeder Teilnehmer muss vorab sein dann allerdings nicht mehr änderbares Passwort dem super-user oder sysadmin melden.

( ) Die Vorbelegung ist nicht möglich und nicht nötig, da zunächst immer die leere Zeichenkette eingetragen ist.

Aufgabe 8

Beurteilen Sie die folgende Aussage. Ein Kommando berücksichtigt immer, ob seine Standardausgabe (stdout) mit einem anderen Prozeß mittels Pipe, mit einer Datei mittels Umlenkung „>“ oder mit einem Gerät (z.B. Terminal) verbunden ist.

( ) Das ist uneingeschränkt richtig.

( ) Das ist uneingeschränkt für alle Kommandos richtig, aber die entsprechenden Optionen müssen gesetzt sein.

( ) Das gilt nur für Shell built-in Kommandos.

( ) Generell ist das nicht so, aber manche Kommandos können das berücksichtigen.

Aufgabe 9

Wir haben eine Datei `bib.xml` zu einer Datei `bib2.xml` verändert. Betrachten Sie die folgenden Kommandos und ihre Ausgaben.

```
cmp bib.xml bib2.xml
bib.xml bib2.xml differieren: Byte 600, Zeile 19.
wegner@rapunzel:~$ diff bib.xml bib2.xml
19d18
< <author><last>Buneman</last><first>Peter</first></author>
wegner@rapunzel:~$ cmp bib.xml bib.xml
wegner@rapunzel:~$ diff bib2.xml bib.xml
18a19
> <author><last>Buneman</last><first>Peter</first></author>
wegner@rapunzel:~$ cmp bib2.xml bib.xml
bib2.xml bib.xml differieren: _____.
```

(a) Worin bestand unsere Änderung?

(b) Ergänzen Sie die Ausgabe des letzten `cmp`-Kommandos!

Aufgabe 10

Das Kommando `which` mit Argument *cmd* liefert den vollen Pfadnamen des Kommandos *cmd*. Also liefert `which passwd` auf unserem Linux-System die Ausgabe

( ) `/usr/bin/passwd`

( ) `/etc/passwd`

( ) `$PATH`

( ) `/usr/bin/which`

Aufgabe 11

Erläutern Sie die Wirkung des unten stehenden Shell-Skripts `nachgefragt`, z.B. anhand des Aufrufs `rm ‘nachgefragt *‘` .

```
#!/bin/bash

for arg in $*
do  echo -n "Pick \"$arg\"? (y/n) " >&2
    read x
    case $x
        in ([yY]*)  echo "$arg"
    esac
done
```

Antwort:

Aufgabe 12

Schreiben Sie ein Shell-Skript `stotter`, das jedes Wort der Kommandozeile doppelt (mit einem Leerzeichen dazwischen) ausgibt, d.h.

```
./stotter Das war aber mal leicht
```

liefert

```
Das Das war war aber aber mal mal leicht leicht
```

ENDE DER KLAUSUR

Prof. Dr. Lutz Wegner
Dipl.-Math. Dipl.-Inf. Sebastian Pape **27. September 2007**

Einführung in UNIX
Klausur zum Sommersemester 2007

MUSTERLÖSUNG

Name:.....Vorname:.....

Matr. Nr.:.....Studiengang:.....

Bearbeiten Sie alle Fragen! Bei Ankreuzaufgaben kann mehr als eine Antwort richtig sein. Hilfsmittel sind nicht zugelassen! Falls Sie für die Beantwortung der Aufgaben zusätzlichen Platz benötigen, verwenden Sie die Rückseiten.

Aufgabe	Punkte max.	Punkte erreicht
1	2	
2	2	
3	2	
4	2	
5	2	
6	2	
7	2	
8	2	
9	4	
10	2	
11	4	
12	4	
Summe	**30**	

Aufgabe 1

Als versteckte Dateien sind nur zugelassen `.`, `..` und `.profile`.

( ) Ja.

(X) Nein, jeder Dateiname, der mit einem Punkt beginnt, bezeichnet eine versteckte Datei.

( ) Nein, aber nur der super-user darf heutzutage solche versteckten Dateien anlegen.

( ) Hängt von der UNIX/LINUX Variante (Distribution) ab.

Aufgabe 2

Damit man mit einem Shell-Skript eine Umgebungsvariable mit einer Zeile der Art `export LANG=de_DE.iso885915@euro` setzen kann, ...

(X) ... muß das Skript mit dem `source`-Kommando (`.`) ausgeführt werden.

( ) ... darf das Skript nicht mit dem `source`-Kommando (`.`) ausgeführt werden.

( ) ... muß `LANG` eine built-in Variable der Shell sein.

( ) ... muß für das Skript das set-user-id Bit gesetzt sein.

Aufgabe 3

In unserer Sammlung gefährlicher und dummer Dateinamen betrachten wir heute den Namen, der nur aus dem Fragezeichen besteht. Kann man ihn erzeugen?

(X) Ja, kann man z.B. mit `touch \?` erzeugen.

(X) Ja, kann man z.B. mit `touch "?"` erzeugen.

(X) Ja, kann man sogar ganz locker mit `touch ?` erzeugen, allerdings nur, wenn es nicht schon Dateinamen bestehend aus einem Zeichen gibt.

( ) Ja, kann man sogar ganz locker mit `touch ?` erzeugen, auch dann, wenn es schon Dateinamen bestehend aus einem Zeichen gibt.

( ) Nein, `?` ist ein Shell-Metazeichen (Joker) und damit verboten.

Aufgabe 4

Unabhängig von allen möglichen Optionen für das `ps`-Kommando sieht man nie

( ) `ps` selbst.

( ) den `init` Prozess.

( ) Hintergrundprozesse.

( ) einen Prozess, der schon `wait()` aufgerufen hat.

( ) einen Zombie-Prozess.

(X) schon vollständig abgeschlossene Prozesse, die in der Vergangenheit liefen.

Aufgabe 5

Die Dateirechte einer Datei werden wo gespeichert?

( ) Im Verzeichnis, in dem der Dateiname eingetragen ist.

(X) Im i-Knoten (inode) der Datei.

( ) In den ersten zwei Bytes des Datenbereichs (des 1. Datenblocks) der Datei.

( ) In der `ar`-Tabelle (access rights table) in `/dev`.

( ) Hängt davon ab, wie die Datei erzeugt wurde, z.B. ob die Datei durch einen hard-link erzeugt wurde.

Aufgabe 6

Ein Benutzer legt eine Datei `x` an, z.B. mit `touch x`. Sie habe danach per default die Rechte 644. Dann legt er den Dateinamen `y` als harten Link auf `x` mit `ln x y` an. Zuletzt führt er `chmod 755 x` aus. Welche Aussage ist richtig?

(X) `y` war zunächst auch 644 und ist jetzt 755.

( ) `y` war zunächst 644 und bleibt 644.

( ) `y` hat zunächst die Linkrechte (default 400) und behält diese immer.

( ) Keine generelle Aussage möglich, da abhängig von der `umask`-Einstellung.

( ) Geht generell nicht, da Dateirechte für Dateien mit Linkzähler >1 nicht änderbar sind.

Aufgabe 7

Wie initialisiert der Administrator oder super-user eines UNIX-Rechners die Passwörter neuer Teilnehmer auf sichere Weise, bis Teilnehmer diese dann selbst ändern?

( ) Üblich sind einheitliche Vorbelegungen wie `anfang` oder `start`.

(X) Üblich sind individuelle Vorbelegungen mit zufällig ausgewählten Passwörtern, die den neuen Teilnehmern schriftlich mitgeteilt werden.

( ) Jeder Teilnehmer muss vorab sein dann allerdings nicht mehr änderbares Passwort dem super-user oder sysadmin melden.

( ) Die Vorbelegung ist nicht möglich und nicht nötig, da zunächst immer die leere Zeichenkette eingetragen ist.

Aufgabe 8

Beurteilen Sie die folgende Aussage. Ein Kommando berücksichtigt immer, ob seine Standardausgabe (stdout) mit einem anderen Prozeß mittels Pipe, mit einer Datei mittels Umlenkung „>“ oder mit einem Gerät (z.B. Terminal) verbunden ist.

( ) Das ist uneingeschränkt richtig.

( ) Das ist uneingeschränkt für alle Kommandos richtig, aber die entsprechenden Optionen müssen gesetzt sein.

( ) Das gilt nur für Shell built-in Kommandos.

(X) Generell ist das nicht so, aber manche Kommandos können das berücksichtigen.

Aufgabe 9

Wir haben eine Datei `bib.xml` zu einer Datei `bib2.xml` verändert. Betrachten Sie die folgenden Kommandos und ihre Ausgaben.

```
cmp bib.xml bib2.xml
bib.xml bib2.xml differieren: Byte 600, Zeile 19.
wegner@rapunzel:~$ diff bib.xml bib2.xml
19d18
< <author><last>Buneman</last><first>Peter</first></author>
wegner@rapunzel:~$ cmp bib.xml bib.xml
wegner@rapunzel:~$ diff bib2.xml bib.xml
18a19
> <author><last>Buneman</last><first>Peter</first></author>
wegner@rapunzel:~$ cmp bib2.xml bib.xml
bib2.xml bib.xml differieren: Byte 600, Zeile 19.
```

(a) Worin bestand unsere Änderung?

Wir haben die 19. Zeile mit <author> ... </author> in bib.xml gelöscht das dann zu bib2.xml wurde.

(b) Ergänzen Sie die Ausgabe des letzten `cmp`-Kommandos!

Aufgabe 10

Das Kommando `which` mit Argument *cmd* liefert den vollen Pfadnamen des Kommandos *cmd*. Also liefert `which passwd` auf unserem Linux-System die Ausgabe

(X) `/usr/bin/passwd`

( ) `/etc/passwd`

( ) `$PATH`

( ) `/usr/bin/which`

Aufgabe 11

Erläutern Sie die Wirkung des unten stehenden Shell-Skripts `nachgefragt`, z.B. anhand des Aufrufs `rm 'nachgefragt *'`.

```
#!/bin/bash

for arg in $*
do  echo -n "Pick \"$arg\"? (y/n) " >&2
    read x
    case $x
        in ([yY]*)  echo "$arg"
    esac
done
```

Antwort:

Das Skript fragt über die Standardfehlerausgabe für jedes Argument aus dem Aufruf, ob es ausgewählt werden soll (picked). Bei Eingabe einer Antwort, die mit y oder Y anfängt, wird das Argument auf die Standardausgabe ausgegeben, sonst wird damit nichts gemacht. Die Wirkung im Beispiel ist wie bei rm -i, d. h. es werden nur die Dateien gelöscht, die bestätigt wurden.

Aufgabe 12

Schreiben Sie ein Shell-Skript `stotter`, das jedes Wort der Kommandozeile doppelt (mit einem Leerzeichen dazwischen) ausgibt, d.h.

```
./stotter Das war aber mal leicht
```

liefert

```
Das Das war war aber aber mal mal leicht leicht
```

```
for i in $*
do
    echo -n "$i $i "
done
echo
```

Hinweis: unsere Lösung erzeugt zwar ein Leerzeichen zu viel, ist aber so ok und kurz.

ENDE DER KLAUSUR

Prof. Dr. Lutz Wegner
Dipl.-Math. Dipl.-Inf. Sebastian Pape **29. Oktober 2007**

Einführung in UNIX
Klausur zum Wintersemester 2007/08

Name:.....Vorname:.....

Matr. Nr.:.....Studiengang:.....

Bearbeiten Sie alle Fragen! Bei Ankreuzaufgaben kann mehr als eine Antwort richtig sein. Hilfsmittel sind nicht zugelassen! Falls Sie für die Beantwortung der Aufgaben zusätzlichen Platz benötigen, verwenden Sie die Rückseiten.

Aufgabe	Punkte max.	Punkte erreicht
1	2	
2	2	
3	2	
4	2	
5	2	
6	2	
7	2	
8	2	
9	2	
10	2	
11	2	
12	2	
13	3	
14	3	
Summe	**30**	

Aufgabe 1

Ergänzen Sie die beiden fehlenden Ausgaben!

```
wegner@rapunzel:~$ echo "-n echo passwd"

_____________________________________________

wegner@rapunzel:~$ echo -n echo passwd

_____________________________________________
```

Aufgabe 2

Auf der Web-Seite http://sunsite.utk.edu/UNIX-help/quickref.html wird behauptet, der Unterschied zwischen `diff` und `cmp` sei der folgende:

```
diff file1 file2     line by line comparison
cmp file1 file2      byte by byte comparison
```

Was ist Ihrer Meinung nach richtig?

( ) `diff` zeigt zeilenweise, wie *file1* geändert werden muß, damit *file2* entsteht.

( ) `diff` ohne weitere Optionen hält nach der ersten entdeckten Unterschiedszeile an.

( ) `cmp` ohne weitere Optionen zeigt die erste (Byte-)Stelle, an der sich *file1* von *file2* unterscheidet, wenn dies der Fall ist.

( ) `cmp` ohne weitere Optionen zeigt alle (Byte-)Stellen, an denen sich *file1* von *file2* unterscheidet, wenn dies der Fall ist.

Aufgabe 3

Welche der folgenden Kommandozeilen liefert den absoluten Pfadnamen des Arbeitsverzeichnisses?

( ) `pwd`

( ) `echo 'pwd'`

( ) `ls -d ‘pwd‘`

( ) `ls -d ~`

Aufgabe 4

Welche Eingaben sind möglich, damit `fix` an Teilnehmer `jr` den Einladungstext `fbausflug` mit dem in der Vorlesung besprochenen `mail`-Kommando verschicken kann?

( ) `mail jr fbausflug`

( ) `mail jr <fbausflug`

( ) `cat fbausflug | mail jr`

( ) Geht nicht, da dieses `mail`-Kommando keine Anhänge kennt.

Aufgabe 5

In unserer Sammlung gefährlicher und dummer Dateinamen betrachten wir heute den Namen, der aus der öffnenden und schließenden eckigen Klammer besteht. Kann man ihn erzeugen?

( ) Ja, kann man z.B. mit `touch \[\]` erzeugen.

( ) Ja, kann man z.B. mit `touch "[]"` erzeugen.

( ) Ja, kann man sogar ganz locker mit `touch []` erzeugen.

( ) Ja, kann man mit `touch '['']'` erzeugen, aber ohne Leerzeichen dazwischen.

( ) Nein, die beiden Klammern `[]` bilden zusammen ein Shell-Metazeichen (Joker) und sind damit verboten.

Aufgabe 6

Die Login-Namen aller an einem Rechner bekannten User erhält man mit

( ) `who`

( ) `id`

( ) `ps -u`

( ) `who are they`

( ) keinem der obigen Kommandos, sicher aber über `/etc/passwd`.

Aufgabe 7

Auf der schon vorher zitierten Web-Seite wird `chmod` wie folgt erklärt.

`chmod -R` *mode dir* (changes all files in *dir*)

Die `man`-Page für `chmod` spricht allerdings von einer rekursiven Änderung aller Dateirechte. Unter Kenntnis der `-R` Option für z. B. `rm` oder `ls`, was ist wohl richtig?

( ) Änderung der Rechte für alle Normaldateien in *`dir`* außer der Rechte von in *`dir`* enthaltenen Verzeichnissen.

( ) Änderung der Rechte für alle Dateien (Normaldateien und Verzeichnisse) in *`dir`*, aber nicht in Unterverzeichnissen.

( ) Änderung der Rechte für alle Dateien (Normaldateien und Verzeichnisse) in *`dir`*, einschließlich aller Dateien in den Unterverzeichnissen.

( ) Eine Option `-R` kann es für `chmod` grundsätzlich nicht geben, weil man die Rechte jeder Datei einzeln und individuell setzen muß.

Aufgabe 8

Benutzer `fix` hat normalerweise `gid=50 (staff)`. Eine von ihm angelegte Datei `x` hat entsprechend den Gruppenbesitzer `staff`. Wenn er mit `newgrp` temporär die neue Gruppenidentität `100 (users)` annimmt an und danach mit `ln x xlink` einen harten Verweis auf `x` sowie mit `cp x xcopy` eine Kopie von `x` erzeugt, dann hat

( ) `xlink` trotzdem den alten Gruppenbesitzer `staff`.

( ) `xlink` den neuen Gruppenbesitzer `users`.

( ) `xcopy` den neuen Gruppenbesitzer `users`.

( ) `xcopy` trotzdem den alten Gruppenbesitzer `staff`.

( ) `xlink` und `xcopy` beide `root` als Gruppenbesitzer, weil Gruppenbesitzwechsel über `newgrp` inzwischen verboten ist.

Aufgabe 9

Die Ausgabe von `ls | wc` ergibt im Heimatverzeichnis von Teilnehmer `wegner` die Ausgabe

```
    10      12      83
```

Warum differieren die Zeilen- und die Wörterzählung?

( ) Liegt an den versteckten Dateien `.` und `..` und ist ganz normal.

( ) Ist eher ungewöhnlich und könnte an einem Dateinamen mit Leerstellen liegen.

( ) Hängt mit der Eigenart von Linux zusammen, die `ls`-Ausgabe spaltenweise aufzubereiten.

( ) Hängt mit der Eigenart von Linux zusammen, die Zeile „`zusammen` *`xyz`*“ an die Ausgabe von `ls` anzuhängen, wobei `xyz` eine Zählung der belegten Blöcke ist.

( ) Hängt mit der Eigenart von Herrn Wegner zusammen, statt ganz korrekt `ls .` (also mit Argument „`.`“) nur einfach `ls` ohne Argument aufzurufen.

Aufgabe 10

Unter Unix gilt die Regel „`init` erbt alle Waisenkinder“. Ein „Waisenkind“ ist ein Prozess, ...

( ) dessen Vaterprozess nicht mehr existiert.

( ) dessen Vaterprozess noch kein `wait` gemacht hat.

( ) der durch `fork` ohne folgendes `exec` entsteht.

( ) der ohne `fork` durch Überlagerung (`exec`) der aktuellen Shell entsteht.

( ) den man `root` mittels `chown` übertragen wollte, was aber inzwischen verboten ist.

Aufgabe 11

Im Kurs haben wir `/dev/null` den großen „Biteimer“ genannt, in den man z. B. Fehlerausgaben umleitet, um lästigen Output zu vernichten. Tatsächlich sind bei `/dev/null` auch Leserechte eingetragen.

( ) Kann nicht sein.

( ) Ja, liefert aber sofort EOT (End of file).

( ) Ja, liefert aber eine Fehlermeldung.

( ) Ja, damit kann man sich die letzten Fehlermeldungen zurückholen (Größe abhängig von Einstellungen in `.bash_history`).

Aufgabe 12

Abmelden eines Teilnehmers geht in Unix oft mit `logoff`, `logout`, `exit`. Im Kurs wird das klassische CTRL-d empfohlen. Was ist dabei zu beachten?

( ) Hat man aus einer Shell eine weitere aufgerufen, wird nur die letzte Shell zugemacht.

( ) Funktioniert nicht, wenn man noch Hintergrundprozesse laufen hat.

( ) Funktioniert nur, wenn man `root` ist.

( ) Abhängig von der eingestellten `sync`-Zeit sollte man vor dem Abmelden warten, bis alle Puffer zurückgeschrieben sind.

( ) Funktioniert nicht am ersten Dienstag im Monat (patch day).

Aufgabe 13

Schreiben Sie ein Shell-Skript ausgezaehlt, das seine Argumente zeilenweise ausgibt. Jeder Zeile ist die Anzahl der noch verbleibenden Argumente und ein Doppelpunkt mit Leerzeichen vorangestellt, wie aus dem Beispielaufruf unten zu sehen.

```
wegner@rapunzel:~/bin$ ./ausgezaehlt ene mene muh
3: ene
2: mene
1: muh
wegner@rapunzel:~/bin$
```

Aufgabe 14:

Ergänzen Sie die Lücken im folgenden Shell-Skript `hardcount`! Das Skript soll alle Normaldateien mit einem Linkzähler größer als 1 in langer Form ausgeben. Eine Beispielausgabe ist unten angegeben.

```
#!/bin/bash
for name in `ls $*`
do
  if [ ! -d ____________________ ]
  then
    set -- `_____________________________`
    if [ $2 -gt 1 ]
    then
        ___________________________________________
    fi
  fi
done
```

Der Inhalt von ~/`bin` und die Ausgabe des Aufrufs von `hardcount .` sei wie folgt.

```
wegner@rapunzel:~/bin$ ls -l
-rwxr-xr-x 1 wegner staff   52 2007-10-23 10:21 ausgezaehlt
drwxr-xr-x 2 wegner staff 4096 2007-10-23 11:33 dummy
-rw-r--r-- 2 wegner staff    0 2007-10-12 12:00 fee
-rw-r--r-- 1 wegner staff    0 2007-10-12 12:00 foo
-rw-r--r-- 3 wegner staff    0 2007-10-12 11:59 gabi
-rwxr-xr-x 1 wegner staff  151 2007-10-23 11:38 hardcount
-rw-r--r-- 3 wegner staff    0 2007-10-12 11:59 rolfi
-rw-r--r-- 3 wegner staff    0 2007-10-12 11:59 susi
-rwxr-xr-x 1 wegner staff   35 2007-10-12 10:47 uhr
wegner@rapunzel:~/bin$ hardcount .
-rw-r--r-- 2 wegner staff 0 2007-10-12 12:00 fee
-rw-r--r-- 3 wegner staff 0 2007-10-12 11:59 gabi
-rw-r--r-- 3 wegner staff 0 2007-10-12 11:59 rolfi
-rw-r--r-- 3 wegner staff 0 2007-10-12 11:59 susi
wegner@rapunzel:~/bin$
```

ENDE DER KLAUSUR

Prof. Dr. Lutz Wegner
Dipl.-Math. Dipl.-Inf. Sebastian Pape **29. Oktober 2007**

Einführung in UNIX
Klausur zum Wintersemester 2007/08

MUSTERLÖSUNG

Name:.....Vorname:.....

Matr. Nr.:.....Studiengang:.....

Bearbeiten Sie alle Fragen! Bei Ankreuzaufgaben kann mehr als eine Antwort richtig sein. Hilfsmittel sind nicht zugelassen! Falls Sie für die Beantwortung der Aufgaben zusätzlichen Platz benötigen, verwenden Sie die Rückseiten.

Aufgabe	Punkte max.	Punkte erreicht
1	2	
2	2	
3	2	
4	2	
5	2	
6	2	
7	2	
8	2	
9	2	
10	2	
11	2	
12	2	
13	3	
14	3	
Summe	**30**	

Aufgabe 1

Ergänzen Sie die beiden fehlenden Ausgaben!

```
wegner@rapunzel:~$ echo "-n echo passwd"
-n echo passwd
wegner@rapunzel:~$ echo -n echo passwd
echo passwdwegner@rapunzel:~$
```

Auch korrekt gewertet ohne wegner@rapunzel:~$ direkt danach

Aufgabe 2

Auf der Web-Seite http://sunsite.utk.edu/UNIX-help/quickref.html wird behauptet, der Unterschied zwischen `diff` und `cmp` sei der folgende:

```
diff file1 file2     line by line comparison
cmp file1 file2      byte by byte comparison
```

Was ist Ihrer Meinung nach richtig?

(X) `diff` zeigt zeilenweise, wie *file1* geändert werden muß, damit *file2* entsteht.

( ) `diff` ohne weitere Optionen hält nach dem ersten entdeckten Unterschiedszeile an.

(X) `cmp` ohne weitere Optionen zeigt die erste (Byte-)Stelle, an der sich *file1* von *file2* unterscheidet, wenn dies der Fall ist.

( ) `cmp` ohne weitere Optionen zeigt alle (Byte-)Stellen, an denen sich *file1* von *file2* unterscheidet, wenn dies der Fall ist.

Aufgabe 3

Welche der folgenden Kommandozeilen liefert den absoluten Pfadnamen des Arbeitsverzeichnisses?

(X) `pwd`

( ) `echo 'pwd'`

(X) `ls -d ‘pwd‘`

( ) `ls -d ~`

Aufgabe 4

Welche Eingaben sind möglich, damit `fix` an Teilnehmer `jr` den Einladungstext `fbausflug` mit dem in der Vorlesung besprochenen `mail`-Kommando verschicken kann?

( ) `mail jr fbausflug`

(X) `mail jr <fbausflug`

(X) `cat fbausflug | mail jr`

( ) Geht nicht, da dieses `mail`-Kommando keine Anhänge kennt.

Aufgabe 5

In unserer Sammlung gefährlicher und dummer Dateinamen betrachten wir heute den Namen, der aus der öffnenden und schließenden eckigen Klammer besteht. Kann man ihn erzeugen?

(X) Ja, kann man z.B. mit `touch \[\]` erzeugen.

(X) Ja, kann man z.B. mit `touch "[]"` erzeugen.

(X) Ja, kann man sogar ganz locker mit `touch []` erzeugen.

(X) Ja, kann man mit `touch '['']'` erzeugen, aber ohne Leerzeichen dazwischen.

( ) Nein, die beiden Klammern `[]` bilden zusammen ein Shell-Metazeichen (Joker) und sind damit verboten.

Aufgabe 6

Die Login-Namen aller an einem Rechner bekannten User erhält man mit

( ) `who`

( ) `id`

( ) `ps -u`

( ) `who are they`

(X) keinem der obigen Kommandos, sicher aber über `/etc/passwd`.

Aufgabe 7

Auf der schon vorher zitierten Web-Seite wird `chmod` wie folgt erklärt.

`chmod -R` *mode dir* (changes all files in *dir*)

Die `man`-Page für `chmod` spricht allerdings von einer rekursiven Änderung aller Dateirechte. Unter Kenntnis der `-R` Option für z. B. `rm` oder `ls`, was ist wohl richtig?

( ) Änderung der Rechte für alle Normaldateien in *`dir`* außer der Rechte von in *`dir`* enthaltenen Verzeichnissen.

( ) Änderung der Rechte für alle Dateien (Normaldateien und Verzeichnisse) in *`dir`*, aber nicht in Unterverzeichnissen.

(X) Änderung der Rechte für alle Dateien (Normaldateien und Verzeichnisse) in *`dir`*, einschließlich aller Dateien in den Unterverzeichnissen.

( ) Eine Option `-R` kann es für `chmod` grundsätzlich nicht geben, weil man die Rechte jeder Datei einzeln und individuell setzen muß.

Aufgabe 8

Benutzer `fix` hat normalerweise `gid=50 (staff)`. Eine von ihm angelegte Datei `x` hat entsprechend den Gruppenbesitzer `staff`. Wenn er mit `newgrp` temporär die neue Gruppenidentität `100 (users)` annimmt an und danach mit `ln x xlink` einen harten Verweis auf `x` sowie mit `cp x xcopy` eine Kopie von `x` erzeugt, dann hat

(X) `xlink` trotzdem den alten Gruppenbesitzer `staff`.

( ) `xlink` den neuen Gruppenbesitzer `users`.

(X) `xcopy` den neuen Gruppenbesitzer `users`.

Hier war in der gedruckten Version ein Fehler (user statt users), der bekanntgegeben wurde.

( ) `xcopy` trotzdem den alten Gruppenbesitzer `staff`.

( ) `xlink` und `xcopy` beide `root` als Gruppenbesitzer, weil Gruppenbesitzwechsel über `newgrp` inzwischen verboten ist.

Aufgabe 9

Die Ausgabe von `ls | wc` ergibt im Heimatverzeichnis von Teilnehmer `wegner` die Ausgabe

```
    10      12      83
```

Warum differieren die Zeilen- und die Wörterzählung?

( ) Liegt an den versteckten Dateien `.` und `..` und ist ganz normal.

(X) Ist eher ungewöhnlich und könnte an einem Dateinamen mit Leerstellen liegen.

( ) Hängt mit der Eigenart von Linux zusammen, die `ls`-Ausgabe spaltenweise aufzubereiten.

( ) Hängt mit der Eigenart von Linux zusammen, die Zeile „`zusammen` *`xyz`*“ an die Ausgabe von `ls` anzuhängen, wobei `xyz` eine Zählung der belegten Blöcke ist.

( ) Hängt mit der Eigenart von Herrn Wegner zusammen, statt ganz korrekt `ls .` (also mit Argument „`.`“) nur einfach `ls` ohne Argument aufzurufen.

Aufgabe 10

Unter Unix gilt die Regel „`init` erbt alle Waisenkinder“. Ein „Waisenkind“ ist ein Prozess, ...

(X) dessen Vaterprozess nicht mehr existiert.

( ) dessen Vaterprozess noch kein `wait` gemacht hat.

( ) der durch `fork` ohne folgendes `exec` entsteht.

( ) der ohne `fork` durch Überlagerung (`exec`) der aktuellen Shell entsteht.

( ) den man `root` mittels `chown` übertragen wollte, was aber inzwischen verboten ist.

Aufgabe 11

Im Kurs haben wir `/dev/null` den großen „Biteimer“ genannt, in den man z. B. Fehlerausgaben umleitet, um lästigen Output zu vernichten. Tatsächlich sind bei `/dev/null` auch Leserechte eingetragen.

( ) Kann nicht sein.

(X) Ja, liefert aber sofort EOT (End of file).

( ) Ja, liefert aber eine Fehlermeldung.

( ) Ja, damit kann man sich die letzten Fehlermeldungen zurückholen (Größe abhängig von Einstellungen in `.bash_history`).

Aufgabe 12

Abmelden eines Teilnehmers geht in Unix oft mit `logoff`, `logout`, `exit`. Im Kurs wird das klassische CTRL-d empfohlen. Was ist dabei zu beachten?

(X) Hat man aus einer Shell eine weitere aufgerufen, wird nur die letzte Shell zugemacht.

( ) Funktioniert nicht, wenn man noch Hintergrundprozesse laufen hat.

( ) Funktioniert nur, wenn man `root` ist.

( ) Abhängig von der eingestellten `sync`-Zeit sollte man vor dem Abmelden warten, bis alle Puffer zurückgeschrieben sind.

( ) Funktioniert nicht am ersten Dienstag im Monat (patch day).

Aufgabe 13

Schreiben Sie ein Shell-Skript ausgezaehlt, das seine Argumente zeilenweise ausgibt. Jeder Zeile ist die Anzahl der noch verbleibenden Argumente und ein Doppelpunkt mit Leerzeichen vorangestellt, wie aus dem Beispielaufruf unten zu sehen.

```
wegner@rapunzel:~/bin$ ./ausgezaehlt ene mene muh
3: ene
2: mene
1: muh
wegner@rapunzel:~/bin$
```

```
for arg
do
    echo $#:" "$arg
    shift
done
```

Aufgabe 14:

Ergänzen Sie die Lücken im folgenden Shell-Skript `hardcount`! Das Skript soll alle Normaldateien mit einem Linkzähler größer als 1 in langer Form ausgeben. Eine Beispielausgabe ist unten angegeben.

```
#!/bin/bash
for name in `ls $*`
do
  if [ ! -d _____$name_____ ]
  then
    set -- `_______ls -l $name_______`
    if [ $2 -gt 1 ]
    then
        _________________ls -l $name_____________________________
    fi
  fi
done
```

Der Inhalt von ~/`bin` und die Ausgabe des Aufrufs von `hardcount .` sei wie folgt.

```
wegner@rapunzel:~/bin$ ls -l
-rwxr-xr-x 1 wegner staff   52 2007-10-23 10:21 ausgezaehlt
drwxr-xr-x 2 wegner staff 4096 2007-10-23 11:33 dummy
-rw-r--r-- 2 wegner staff    0 2007-10-12 12:00 fee
-rw-r--r-- 1 wegner staff    0 2007-10-12 12:00 foo
-rw-r--r-- 3 wegner staff    0 2007-10-12 11:59 gabi
-rwxr-xr-x 1 wegner staff  151 2007-10-23 11:38 hardcount
-rw-r--r-- 3 wegner staff    0 2007-10-12 11:59 rolfi
-rw-r--r-- 3 wegner staff    0 2007-10-12 11:59 susi
-rwxr-xr-x 1 wegner staff   35 2007-10-12 10:47 uhr
wegner@rapunzel:~/bin$ hardcount .
-rw-r--r-- 2 wegner staff 0 2007-10-12 12:00 fee
-rw-r--r-- 3 wegner staff 0 2007-10-12 11:59 gabi
-rw-r--r-- 3 wegner staff 0 2007-10-12 11:59 rolfi
-rw-r--r-- 3 wegner staff 0 2007-10-12 11:59 susi
wegner@rapunzel:~/bin$
```

ENDE DER KLAUSUR

Prof. Dr. Lutz Wegner
Dipl.-Math Dipl. Inf. Sebastian Pape **10. Juli 2008**

Einführung in UNIX
Klausur zum Sommersemester 2008

Name:.....Vorname:.....

Matr. Nr.:.....Studiengang:.....

Bearbeiten Sie alle Fragen! Bei Ankreuzaufgaben kann mehr als eine Antwort richtig sein. Hilfsmittel sind nicht zugelassen! Falls Sie für die Beantwortung der Aufgaben zusätzlichen Platz benötigen, verwenden Sie die Rückseiten oder separate Blätter mit Ihrem Namen.

Aufgabe	Punkte max.	Punkte erreicht
1	2	
2	2	
3	2	
4	2	
5	2	
6	2	
7	2	
8	2	
9	2	
10	2	
11	2	
12	3	
Summe	**25**	

Aufgabe 1

Ein Anwender hat eine Datei `crypttest` mit dem Inhalt „Hallo Leute“. Er möchte durch zweifache Anwendung des `crypt`-Kommandos die Datei doppelt verschlüsseln (gemäß dem alten Motto: doppelt hält besser). `crypt` unter Linux macht sowohl Ver- als auch die Entschlüsselung und verwendet ein symetrisches Verfahren (gleicher Schlüssel für Ver- und Entschlüsselung). Was ist richtig?

( ) Zweifach verschlüsseln ginge mit `crypt <crypttest >ctest;`
`crypt <ctest >cctest` und Eingabe von zweimal dem selben Passwort.

( ) Zweifach verschlüsseln ginge mit `crypt <crypttest >ctest;`
`crypt <ctest >cctest` aber Eingabe zweier unterschiedlicher Passwörter.

( ) Ginge mit `crypt -`*erstesPasswort* `<crypttest >ctest;`
`crypt -`*zweitesPasswort* `<ctest >cctest`
und zweimal dem selben Passwort an den angegebenen Stellen des Kommandoaufrufs.

( ) Ginge mit `crypt -`*erstesPasswort* `<crypttest >ctest;`
`crypt -`*zweitesPasswort* `<ctest >cctest` und zwei unterschiedlichen Passwörtern an den angegebenen Stellen des Kommandoaufrufs.

( ) Zweifach verschlüsseln mit `crypt` geht grundsätzlich nicht.

Aufgabe 2

Die Datei `cities` enthält einige Städtenamen wie gezeigt.

```
wegner@rapunzel:~/klausur$ more cities
OSLO
Rom
Paris
oslo
ROM
wegner@rapunzel:~/klausur$ tr 'a-z' 'A-Z' <cities | sort | uniq
OSLO
PARIS
ROM
wegner@rapunzel:~/klausur$ sort <cities | uniq -i
oslo
Paris
Rom
wegner@rapunzel:~/klausur$
```

Erläutern Sie auf der Rückseite die unterschiedliche Ausgabe der unteren zwei Kommandos.

Aufgabe 3

Das `newgrp`-Kommando hat das SUID-Bit gesetzt, wie gezeigt. Was macht `newgrp` und wer leiht hier wem welche Berechtigung?

```
-rwsr-xr-x 1 root root 20056 2007-02-27 08:53 /usr/bin/newgrp
```

Aufgabe 4

Die Ausgabe von `ls -l` enthält unter Linux eine Angabe zum belegten Speicherplatz (hier z.B. insgesamt 8 Blöcke zu je 4KB). Erläutern Sie auf der Rückseite der gegenüberliegenden Seite die Ausgabe der Größen unten nach den Kommandos `touch` und `ln`. Warum erscheint die Zählung der Größen auf den ersten Blick falsch?

```
wegner@rapunzel:~/klausur$ ls -l
insgesamt 8
-rw-r--r-- 1 wegner staff 24 2008-07-08 15:07 cities
-rw-r--r-- 1 wegner staff 12 2008-07-08 14:24 ctest
wegner@rapunzel:~/klausur$ touch leer
wegner@rapunzel:~/klausur$ ls -l
insgesamt 8
-rw-r--r-- 1 wegner staff 24 2008-07-08 15:07 cities
-rw-r--r-- 1 wegner staff 12 2008-07-08 14:24 ctest
-rw-r--r-- 1 wegner staff  0 2008-07-09 11:59 leer
wegner@rapunzel:~/klausur$ ln ctest clink
wegner@rapunzel:~/klausur$ ls -l
insgesamt 12
-rw-r--r-- 1 wegner staff 24 2008-07-08 15:07 cities
-rw-r--r-- 2 wegner staff 12 2008-07-08 14:24 clink
-rw-r--r-- 2 wegner staff 12 2008-07-08 14:24 ctest
-rw-r--r-- 1 wegner staff  0 2008-07-09 11:59 leer
wegner@rapunzel:~/klausur$ ls -la
insgesamt 20
drwx------ 2 wegner staff 4096 2008-07-09 11:59 .
drwxr-xr-x 8 wegner staff 4096 2008-07-08 15:07 ..
-rw-r--r-- 1 wegner staff   24 2008-07-08 15:07 cities
-rw-r--r-- 2 wegner staff   12 2008-07-08 14:24 clink
-rw-r--r-- 2 wegner staff   12 2008-07-08 14:24 ctest
-rw-r--r-- 1 wegner staff    0 2008-07-09 11:59 leer
wegner@rapunzel:~/klausur$
```

Aufgabe 5

Erläutern Sie den Unterschied zwischen einem Prozess, der ein Waisenkind ist, und einem Zombieprozess.

Aufgabe 6

Welche Rolle spielen die Dateideskriptoren 0, 1, 2?

Aufgabe 7:

Warum wird heute - anders als im Skript - nicht mehr empfohlen, das Arbeitsverzeichnis ganz vorne in PATH einzufügen? Kurze Erläuterung.

Aufgabe 8

Was bezeichnet `/dev/tty`? Gibt es diesen Eintrag in `/dev` tatsächlich?

Aufgabe 9

Die Manualseite zu `echo` nennt eine Option `-e`:

```
    -e     enable interpretation of backslash escapes
```

Erläutern Sie die folgenden Ausgaben!

(a)
```
wegner@rapunzel:~/klausur$ echo \n
n
```

(b)
```
wegner@rapunzel:~/klausur$ echo -e \n
n
```

(c)
```
wegner@rapunzel:~/klausur$ echo \\n
\n
```

(d)
```
wegner@rapunzel:~/klausur$ echo -e \\n
```

(e)
```
wegner@rapunzel:~/klausur$ echo -en \\n

wegner@rapunzel:~/klausur$
```

Aufgabe 10

Wäre `read` kein Spezialkommando der Shell (built-in command), hätte man nach dem Aufruf der Kommandozeile `read eingabe` und der Rückkehr zur Shell keinen Wert in der Variablen `eingabe`. Richtig?

( ) Nein, könnte man mit `export` umgehen.

( ) Nein, könnte man mit `import` umgehen.

( ) Das Problem stellt sich nicht, weil `read` nur in Shellskripten auftreten darf.

( ) Das Problem stellt sich nicht, weil `read` als Kommando nicht existiert, sondern nur als Systemaufruf (system call).

( ) Ja, liegt an der Einbahnstraße für die Werteweitergabe.

Aufgabe 11

Für eine Datei sei z.B. der Besitzer `fix` und die Besitzergruppe `stkom` (Studienkommission) eingetragen. Die Dateirechte für `u`, `g`, `o` lassen sich dann von wem neu setzen?

( ) Nur vom Besitzer (`fix`) alle Rechte.

( ) Vom Besitzer alle Rechte, von jedem in der Gruppe (`stkom`) nur die Gruppenrechte.

( ) Von jedem in der Gruppe (`stkom`) alle Rechte unabhängig von der gegenwärtigen Gruppenkennung.

( ) Von jedem in der Gruppe (`stkom`) alle Rechte, wenn vorher die Gruppenkennung mit `newgrp` angenommen wurde.

( ) Hängt vom SGID (set group-owner ID) Bit ab.

( ) Hängt davon ab, ob es eine Normaldatei, Verzeichnis oder Gerät ist.

Aufgabe 12

Am Mittwoch aufgerufen ergab das Shell-Skript `WE` die folgende Ausgabe:

```
wegner@rapunzel:~/klausur$ ./WE
noch 3 Tage bis zum Wochenende!
wegner@rapunzel:~/klausur$
```

Ergänzen Sie das folgende Shell-Skript `WE`!

```
set `__________________`

case ______ in

    Mo) tage=5;;

    Di) tage=4;;

    Mi) tage=3;;

    Do) tage=2;;

    Fr) tage=1;;

    *) tage=0;;

esac

echo noch $tage Tage bis zum Wochenende!
```

ENDE DER KLAUSUR

Prof. Dr. Lutz Wegner
Dipl.-Math Dipl. Inf. Sebastian Pape **10. Juli 2008**

Einführung in UNIX
Klausur zum Sommersemester 2008

MUSTERLÖSUNG

Name:.....Vorname:.....

Matr. Nr.:.....Studiengang:.....

Bearbeiten Sie alle Fragen! Bei Ankreuzaufgaben kann mehr als eine Antwort richtig sein. Hilfsmittel sind nicht zugelassen! Falls Sie für die Beantwortung der Aufgaben zusätzlichen Platz benötigen, verwenden Sie die Rückseiten oder separate Blätter mit Ihrem Namen.

Aufgabe	Punkte max.	Punkte erreicht
1	2	
2	2	
3	2	
4	2	
5	2	
6	2	
7	2	
8	2	
9	2	
10	2	
11	2	
12	3	
Summe	**25**	

Aufgabe 1

Ein Anwender hat eine Datei `crypttest` mit dem Inhalt „Hallo Leute“. Er möchte durch zweifache Anwendung des `crypt`-Kommandos die Datei doppelt verschlüsseln (gemäß dem alten Motto: doppelt hält besser). `crypt` unter Linux macht sowohl Ver- als auch die Entschlüsselung und verwendet ein symetrisches Verfahren (gleicher Schlüssel für Ver- und Entschlüsselung). Was ist richtig?

( ) Zweifach verschlüsseln ginge mit `crypt <crypttest >ctest;`
`crypt <ctest >cctest` und Eingabe von zweimal dem selben Passwort.

(X) Zweifach verschlüsseln ginge mit `crypt <crypttest >ctest;`
`crypt <ctest >cctest` aber Eingabe zweier unterschiedlicher Passwörter.

( ) Ginge mit `crypt -`*erstesPasswort* `<crypttest >ctest;`
`crypt -`*zweitesPasswort* `<ctest >cctest`
und zweimal dem selben Passwort an den angegebenen Stellen des Kommandoaufrufs.

( ) Ginge mit `crypt -`*erstesPasswort* `<crypttest >ctest;`
`crypt -`*zweitesPasswort* `<ctest >cctest` und zwei unterschiedlichen Passwörtern an den angegebenen Stellen des Kommandoaufrufs.

( ) Zweifach verschlüsseln mit `crypt` geht grundsätzlich nicht.

Aufgabe 2

Die Datei `cities` enthält einige Städtenamen wie gezeigt.

```
wegner@rapunzel:~/klausur$ more cities
OSLO
Rom
Paris
oslo
ROM
wegner@rapunzel:~/klausur$ tr 'a-z' 'A-Z' <cities | sort | uniq
OSLO
PARIS
ROM
wegner@rapunzel:~/klausur$ sort <cities | uniq -i
oslo
Paris
Rom
wegner@rapunzel:~/klausur$
```

Erläutern Sie auf der Rückseite die unterschiedliche Ausgabe der unteren zwei Kommandos.

Im ersten Fall wird der Text in Großbuchstaben übersetzt (tr), dann aufsteigend sortiert, dann werden Duplikate entfernt. Im zweiten Fall wird ausgenutzt, dass bei dem hier verwendeten sort Zeilen, die sich nur in Groß- und Kleinschreibung unterscheiden, aufeinander folgen. uniq mit Option -i unterscheidet dann nicht nach Groß- und Kleinschreibung.

Aufgabe 3

Das `newgrp`-Kommando hat das SUID-Bit gesetzt, wie gezeigt. Was macht `newgrp` und wer leiht hier wem welche Berechtigung?

```
-rwsr-xr-x 1 root root 20056 2007-02-27 08:53 /usr/bin/newgrp
```

Root leiht dem Aufrufer das Recht, eine neue Gruppenkennung anzunehmen.

zu Aufgabe 4 unten:

ls addiert die Größen. Bei touch wird eine leere Datei angelegt und kein echter Speicherplatz reserviert. Bei ln kann das System nicht erkennen, dass es von der Datei nur ein physisches Exemplar gibt, der Link kann ja auch in andere Verzeichnisse zeigen. Deshalb zählt der Platz doppelt.

Aufgabe 4

Die Ausgabe von `ls -l` enthält unter Linux eine Angabe zum belegten Speicherplatz (hier z.B. insgesamt 8 Blöcke zu je 4KB). Erläutern Sie auf der Rückseite der gegenüberliegenden Seite die Ausgabe der Größen unten nach den Kommandos `touch` und `ln`. Warum erscheint die Zählung der Größen auf den ersten Blick falsch?

```
wegner@rapunzel:~/klausur$ ls -l
insgesamt 8
-rw-r--r-- 1 wegner staff 24 2008-07-08 15:07 cities
-rw-r--r-- 1 wegner staff 12 2008-07-08 14:24 ctest
wegner@rapunzel:~/klausur$ touch leer
wegner@rapunzel:~/klausur$ ls -l
insgesamt 8
-rw-r--r-- 1 wegner staff 24 2008-07-08 15:07 cities
-rw-r--r-- 1 wegner staff 12 2008-07-08 14:24 ctest
-rw-r--r-- 1 wegner staff  0 2008-07-09 11:59 leer
wegner@rapunzel:~/klausur$ ln ctest clink
wegner@rapunzel:~/klausur$ ls -l
insgesamt 12
-rw-r--r-- 1 wegner staff 24 2008-07-08 15:07 cities
-rw-r--r-- 2 wegner staff 12 2008-07-08 14:24 clink
-rw-r--r-- 2 wegner staff 12 2008-07-08 14:24 ctest
-rw-r--r-- 1 wegner staff  0 2008-07-09 11:59 leer
wegner@rapunzel:~/klausur$ ls -la
insgesamt 20
drwx------ 2 wegner staff 4096 2008-07-09 11:59 .
drwxr-xr-x 8 wegner staff 4096 2008-07-08 15:07 ..
-rw-r--r-- 1 wegner staff   24 2008-07-08 15:07 cities
-rw-r--r-- 2 wegner staff   12 2008-07-08 14:24 clink
-rw-r--r-- 2 wegner staff   12 2008-07-08 14:24 ctest
-rw-r--r-- 1 wegner staff    0 2008-07-09 11:59 leer
wegner@rapunzel:~/klausur$
```

siehe oben unter A3

Aufgabe 5

Erläutern Sie den Unterschied zwischen einem Prozess, der ein Waisenkind ist, und einem Zombieprozess.

Ein „Waisenkind" ist ein Prozess, dessen Vaterprozess gestorben ist und damit kein wait für das Waisenkind machen kann. Der init-Prozess erbt alle Waisenkinder.

Ein „Zombie-Prozess" ist ein Prozess, der fertig abgearbeitet wurde, aber noch darauf wartet, dass der Vater ein wait macht. Er wird schon abgeräumt, bleibt aber noch in der Prozesstabelle mit seinem Rückgabewert. Zombie-Prozesse reagieren nicht auf kill -9 (weil sie schon tot sind).

Aufgabe 6

Welche Rolle spielen die Dateideskriptoren 0, 1, 2?

Entsprechen stdin (Standardeingabe), stdout (~ausgabe), stderr (~fehlerausgabe)

Aufgabe 7:

Warum wird heute - anders als im Skript - nicht mehr empfohlen, das Arbeitsverzeichnis ganz vorne in PATH einzufügen? Kurze Erläuterung.

Aus Sicherheitsgründen. Ein Angreifer könnte in /tmp ein schädliches Programm unter dem Namen eines gängigen Kommandos, z.B. ls, ablegen. Wechselt man nach /tmp und ruft ls auf, wird der Schadcode statt des erwarteten Kommandos /bin/ls ausgeführt.

Aufgabe 8

Was bezeichnet `/dev/tty`? Gibt es diesen Eintrag in `/dev` tatsächlich?

Das eigene Terminal. Den Eintrag gibt es tatsächlich.

Aufgabe 9

Die Manualseite zu `echo` nennt eine Option `-e`:

```
    -e     enable interpretation of backslash escapes
```

Erläutern Sie die folgenden Ausgaben!

(a)
```
wegner@rapunzel:~/klausur$ echo \n
n
```

(b)
```
wegner@rapunzel:~/klausur$ echo -e \n
n
```

(c)
```
wegner@rapunzel:~/klausur$ echo \\n
\n
```

(d)
```
wegner@rapunzel:~/klausur$ echo -e \\n
```

(e)
```
wegner@rapunzel:~/klausur$ echo -en \\n

wegner@rapunzel:~/klausur$
```

(a) Escape-Sequenz wird nicht interpretiert

(b) Escape-Sequenz wird interpretiert, aber Backslash maskiert nur das n

(c) Escape-Sequenz wird nicht interpretiert, \n wirkt deshalb nicht wie newline

(d) Escape-Sequenz wird interpretiert, \n wirkt deshalb als (zusätzliches) newline

(e)wie (d), -n unterdrückt aber das normal newline des echo.

Aufgabe 10

Wäre `read` kein Spezialkommando der Shell (built-in command), hätte man nach dem Aufruf der Kommandozeile `read eingabe` und der Rückkehr zur Shell keinen Wert in der Variablen `eingabe`. Richtig?

( ) Nein, könnte man mit `export` umgehen.

( ) Nein, könnte man mit `import` umgehen.

( ) Das Problem stellt sich nicht, weil `read` nur in Shellskripten auftreten darf.

( ) Das Problem stellt sich nicht, weil `read` als Kommando nicht existiert, sondern nur als Systemaufruf (system call).

(X) Ja, liegt an der Einbahnstraße für die Werteweitergabe.

Aufgabe 11

Für eine Datei sei z.B. der Besitzer `fix` und die Besitzergruppe `stkom` (Studienkommission) eingetragen. Die Dateirechte für `u`, `g`, `o` lassen sich dann von wem neu setzen?

(X) Nur vom Besitzer (`fix`) alle Rechte.

( ) Vom Besitzer alle Rechte, von jedem in der Gruppe (`stkom`) nur die Gruppenrechte.

( ) Von jedem in der Gruppe (`stkom`) alle Rechte unabhängig von der gegenwärtigen Gruppenkennung.

( ) Von jedem in der Gruppe (`stkom`) alle Rechte, wenn vorher die Gruppenkennung mit `newgrp` angenommen wurde.

( ) Hängt vom SGID (set group-owner ID) Bit ab.

( ) Hängt davon ab, ob es eine Normaldatei, Verzeichnis oder Gerät ist.

Aufgabe 12

Am Mittwoch aufgerufen ergab das Shell-Skript `WE` die folgende Ausgabe:

```
wegner@rapunzel:~/klausur$ ./WE
noch 3 Tage bis zum Wochenende!
wegner@rapunzel:~/klausur$
```

Ergänzen Sie das folgende Shell-Skript `WE`!

```
set `___date___`

case __$1__ in

    Mo) tage=5;;

    Di) tage=4;;

    Mi) tage=3;;

    Do) tage=2;;

    Fr) tage=1;;

    *) tage=0;;

esac

echo noch $tage Tage bis zum Wochenende!
```

ENDE DER KLAUSUR

Prof. Dr. Lutz Wegner
Dipl.-Math. Dipl.-Inf. Sebastian Pape **19. Februar 2009**

Einführung in UNIX
Klausur zum Wintersemester 2008/09

Name:.....Vorname:.....

Matr. Nr.:.....Studiengang:.....

Bearbeiten Sie alle Fragen! Bei Ankreuzaufgaben kann mehr als eine Antwort richtig sein. Hilfsmittel sind nicht zugelassen! Verwenden Sie die Rückseiten, falls Sie für die Beantwortung der Aufgaben zusätzlichen Platz benötigen.

Aufgabe	Punkte max.	Punkte erreicht
1	2	
2	2	
3	2	
4	4	
5	2	
6	2	
7	2	
8	2	
9	2	
10	3	
11	2	
12	3	
13	2	
14	5	
15	5	
Summe	**40**	

Aufgabe 1

Welche der folgenden Kommandos erzeugt die Ausgabe

```
hello world
```

auf einer Zeile für sich ohne weitere Ausgaben davor und danach?

( ) `wegner@rapunzel:~$ echo hello world`

( ) `wegner@rapunzel:~$ nice echo hello world`

( ) `wegner@rapunzel:~$ which echo hello world`

( ) `wegner@rapunzel:~$ echo hello world &`

( ) `wegner@rapunzel:~$ echo -n hello world`

( ) `wegner@rapunzel:~$ echo "hello world"`

( ) `wegner@rapunzel:~$ echo 'hello world'`

Aufgabe 2:

Was bewirkt das folgende Kommando?

```
wegner@rapunzel:~$ echo "hello world" >stdout
```

Aufgabe 3:

Kommentieren Sie den Ablauf der folgenden Kommandos und das Verhalten von `cmp` und `diff` bei Verzeichnissen unter LINUX. Verwenden Sie die Blattrückseite. Es genügen ca. 5 Sätze.

```
wegner@rapunzel:~$ mkdir adir bdir
wegner@rapunzel:~$ touch adir/xfile adir/yfile
wegner@rapunzel:~$ touch bdir/zfile bdir/yfile
wegner@rapunzel:~$ diff adir bdir
Nur in adir: xfile.
Nur in bdir: zfile.
wegner@rapunzel:~$ cmp adir bdir
cmp: adir: Ist ein Verzeichnis
wegner@rapunzel:~$
```

Aufgabe 4:

Im Verzeichnis `bin`, das auf meinem Kommandosuchpfad liegt, gibt es eine Datei `myhallo`, die nur eine Zeile `echo $HALLO` enthält und ausführbar ist. `HALLO` ist nicht für den Export markiert.

(a) Was liefert ein einfacher Aufruf mit der Zeile unten?

```
myhallo
```

_____

(b) Was liefert der Aufruf mit der folgenden Zeile?

```
HALLO=world myhallo
```

_____

(c) Nehmen wir an, in `mypwd` würde die Zeile `echo $PWD` stehen.
Erklären Sie die Ausgabe unten! Müßte die Ausgabe nicht `/home` lauten?

```
wegner@rapunzel:~/kurs$ pwd
/home/wegner/kurs
wegner@rapunzel:~/kurs$ PWD=/home mypwd
/home/wegner/kurs
wegner@rapunzel:~/kurs$
```

Aufgabe 5:

Auch wenn man für das Verschlüsseln von emails eher mit PGP arbeiten würde, nehmen wir doch mal an, Sie wollten eine Datei `brief` mittels `crypt` verschlüsseln und den verschlüsselten Text als datei `xbrief` an einen Teilnehmer `buddy` verschicken. Welche der folgenden Aussagen sind richtig?

( ) Die Verschlüsselung könnte mit `crypt <brief >xbrief` unter Eingabe des Schlüssels (passphrase) erfolgen.

( ) Der Versand würde mit `mail buddy <xbrief` erfolgen.

( ) Der Empfänger `buddy` müßte den Schlüssel kennen, sonst kann er den Inhalt nicht im Klartext lesen.

( ) Eine sehr sichere Methode wäre es, den Schlüssel in die Betreff-Zeile zu schreiben.

( ) Geht nicht, da das `mail`-Kommando keine verschlüsselten Texte versenden kann.

Aufgabe 6:

In unserer Sammlung gefährlicher und dummer Dateinamen betrachten wir heute den leeren Namen, der aus null Zeichen besteht. Kann man ihn mit `touch` erzeugen?

( ) Ja, kann man z.B. mit `touch \` erzeugen.

( ) Ja, kann man z.B. mit `touch ""` erzeugen, aber ohne Leerzeichen dazwischen.

( ) Ja, kann man sogar ganz locker mit `touch` erzeugen.

( ) Ja, kann man mit `touch ''` erzeugen, aber ohne Leerzeichen dazwischen.

( ) Ja, kann man mit `touch ‘echo -n‘` erzeugen.

( ) Nein, leere Dateinamen gibt es nicht.

Aufgabe 7:

Wir betrachten die drei Kommandos (a) - (c). Welche der Ausgaben sollten immer gleich sein, unabhängig vom aktuellen Inhalt von `klausur`?

(a)
```
wegner@rapunzel:~/klausur$ ls -ia | cat -
...
```

(b)
```
wegner@rapunzel:~/klausur$ ls -ia | sort
...
```

(c)
```
wegner@rapunzel:~/klausur$ ls -ia | sort -k 2
...
```

Die Ausgaben von _____ sind gleich.

Aufgabe 8:

Ähnlich zum Shell-Skript `cx` aus dem Kurs schreiben wir ein Shell-Skript `xweg` mit der einzigen Zeile

```
chmod -x $*
```

und machen `xweg` ausführbar. Danach rufen wir aber `xweg xweg` auf. Was ist richtig?

( ) Das geht gar nicht, weil wir ja im Skript `xweg` uns selbst das Ausführungsrecht entziehen.

( ) Das geht einmal, aber danach kann man `xweg` nicht mehr als ausführbares Shell-Skript aufrufen.

( ) Das geht gar nicht, weil im Skript `chmod -x *` stehen müßte, nicht `$*`.

( ) Das geht nicht, weil die Shell erkennt, dass das Argument und der Kommandoname identisch sind, was verboten ist.

Aufgabe 9:

Der grundsätzliche Vorteil eines Softlinks (eines weichen Verweises) gegenüber dem Hardlink (harten Verweis) ist, dass der Softlink

( ) über Partitionen (volumes) hinweg funktioniert.

( ) prüft, ob die Zieldatei auch existiert.

( ) den Linkzähler in jedem Fall richtig hochsetzt.

( ) den Rechner sehr viel weniger belastet als ein harter Verweis.

( ) `root` als Besitzer zuläßt, was beim Hardlink aus Sicherheitsgründen verboten ist.

Aufgabe 10:

Ergänzen Sie die fehlende Zeile unten!

```
wegner@rapunzel:~/bin$ mkdir leerdir
wegner@rapunzel:~/bin$ ls leerdir | wc
      0       0       0
wegner@rapunzel:~/bin$ ls leerdir -a | wc
```

_____

Aufgabe 11:

Unter einem Zombie-Prozess, versteht man in UNIX einen Prozess, ...

( ) der zum Ende gekommen ist, aber dessen Vaterprozess nicht mehr existiert.

( ) der zum Ende gekommen ist, dessen Vaterprozess aber noch kein `wait` gemacht hat.

( ) der durch `fork` ohne folgendes `exec` entstanden ist.

( ) der ohne `fork` durch Überlagerung (`exec`) der aktuellen Shell entstand.

( ) den man `root` mittels `chown` übertragen wollte, was aber inzwischen verboten ist.

Aufgabe 12:

In einer Multiuser-Umgebung mit gemeinsam genutztem Drucker ist ein häufiges Problem, dass der Drucker blockiert ist, weil der zugehörige Dämonprozess abgestürzt ist. Wie heißt der Prozess für den Standarddrucker `/dev/lp0`, welche Aufgabe hat er und in welcher Beziehung steht er zu `lpr`? Es reichen 3 - 5 Sätze Antwort auf der Rückseite.

Aufgabe 13:

Von Zeit zu Zeit wird der Sysadmin einen UNIX-Server herunterfahren. Was ist richtig?

( ) „Stecker ziehen“ im laufenden Betrieb ist die Regelmethode, um UNIX-Systeme herunterzufahren.

( ) UNIX-Systeme werden in der Regel mit dem `shutdown`-Befehl heruntergefahren.

( ) UNIX-Systeme werden in der Regel durch `kill -9 pager` heruntergefahren.

( ) Zum Auslösen des `shutdown`-Befehls braucht man `root`-Berechtigung.

( ) Solange mindestens ein Benutzer außer `root` angemeldet ist, läßt sich der Rechner nicht (gegen dessen Willen) herunterfahren.

Aufgabe 14:

Ergänzen Sie die Lücken im folgenden Shell-Skript `mydelete`, das alle Normaldateien oder **leere** Verzeichnisse löscht, die als Argumente übergebenen werden. Natürlich nur, sofern die Rechte dies erlauben.

Hinweis: `test -n` *string* liefert wahr genau dann, wenn die Länge von *string* größer Null ist.

```
#!/bin/bash
while [ -n "________________" ]; do
    if [ -d "____________" ]; then
        ___________ "$1";
    else
        _________ "$1";
    fi
    ____________
done
```

Aufgabe 15:

Der folgende Programmteil soll Aufgabe 14 dahingehend ergänzen, dass bei **nichtleeren** Verzeichnissen gefragt wird, ob das Verzeichnis mit allen Inhalten trotzdem gelöscht werden soll. Wenn die eingelesene Antwort „y“ oder „Y“ ist, dann löschen Sie das Verzeichnis rekursiv.

Hinweis: `read -n` *nchars var* liest nur *nchars* Zeichen, weist sie der Variablen *var* zu und kehrt sofort zurück ohne auf eine komplete Eingabezeile zu warten.

```
# an richtiger Stelle einsetzen bzw. ersetzen
    if [ _______ls "$1"|wc -l______ -eq 0 ]; then
        #directory empty
        _________________ "$1";
    else
        echo "Directory not empty. Delete anyway?"
        # read 1 key
        read -n 1 ____________________
        # clear line
        echo
        if [ "$notempty" = y ] ___ [ "$notempty" = Y ]; then
            ___________________ "$1";
        fi
    fi
# Ende Einsetzung Aufgabe 15
```

ENDE DER KLAUSUR

Prof. Dr. Lutz Wegner
Dipl.-Math. Dipl.-Inf. Sebastian Pape **19. Februar 2009**

Einführung in UNIX
Klausur zum Wintersemester 2008/09

Name:.....MUSTERLÖSUNG..... **Vorname:**.....

Matr. Nr.:.....**Studiengang:**.....

Bearbeiten Sie alle Fragen! Bei Ankreuzaufgaben kann mehr als eine Antwort richtig sein. Hilfsmittel sind nicht zugelassen! Verwenden Sie die Rückseiten, falls Sie für die Beantwortung der Aufgaben zusätzlichen Platz benötigen.

Aufgabe	Punkte max.	Punkte erreicht
1	2	
2	2	
3	2	
4	4	
5	2	
6	2	
7	2	
8	2	
9	2	
10	3	
11	2	
12	3	
13	2	
14	5	
15	5	
Summe	**40**	

Aufgabe 1

Welche der folgenden Kommandos erzeugt die Ausgabe

```
hello world
```

auf einer Zeile für sich ohne weitere Ausgaben davor und danach?

(X) `wegner@rapunzel:~$ echo hello world`

(X) `wegner@rapunzel:~$ nice echo hello world`

( ) `wegner@rapunzel:~$ which echo hello world`

( ) `wegner@rapunzel:~$ echo hello world &`

( ) `wegner@rapunzel:~$ echo -n hello world`

(X) `wegner@rapunzel:~$ echo "hello world"`

(X) `wegner@rapunzel:~$ echo 'hello world'`

Aufgabe 2:

Was bewirkt das folgende Kommando?

```
wegner@rapunzel:~$ echo "hello world" >stdout
```

__ **erzeugt eine Datei stdout mit Inhalt hello world** _____

Aufgabe 3:

Kommentieren Sie den Ablauf der folgenden Kommandos und das Verhalten von `cmp` und `diff` bei Verzeichnissen unter LINUX. Verwenden Sie die Blattrückseite. Es genügen ca. 5 Sätze.

```
wegner@rapunzel:~$ mkdir adir bdir
wegner@rapunzel:~$ touch adir/xfile adir/yfile
wegner@rapunzel:~$ touch bdir/zfile bdir/yfile
wegner@rapunzel:~$ diff adir bdir
Nur in adir: xfile.
Nur in bdir: zfile.
wegner@rapunzel:~$ cmp adir bdir
cmp: adir: Ist ein Verzeichnis
wegner@rapunzel:~$
```

Das Kommando diff vergleicht auch Verzeichnisse und gibt an, was im 2. Verzeichnis (bdir) fehlt, bzw. mehr drin ist, im Vergleich zum 1. Verzeichnis (adir).

Das Kommando cmp kann Verzeichnisse gar nicht vergleichen.

Aufgabe 4:

Im Verzeichnis `bin`, das auf meinem Kommandosuchpfad liegt, gibt es eine Datei `myhallo`, die nur eine Zeile `echo $HALLO` enthält und ausführbar ist. `HALLO` ist nicht für den Export markiert.

(a) Was liefert ein einfacher Aufruf mit der Zeile unten?

```
myhallo
```

___nur eine leere Zeile _____

(b) Was liefert der Aufruf mit der folgenden Zeile?

```
HALLO=world myhallo
```

____eine Zeile mit dem Wort „world" ____

(c) Nehmen wir an, in `mypwd` würde die Zeile `echo $PWD` stehen.
Erklären Sie die Ausgabe unten! Müßte die Ausgabe nicht `/home` lauten?

```
wegner@rapunzel:~/kurs$ pwd
/home/wegner/kurs
wegner@rapunzel:~/kurs$ PWD=/home mypwd
/home/wegner/kurs
wegner@rapunzel:~/kurs$
```

Die interne Variable PWD wird neu gesetzt bei jedem fork der Shell. Dadurch wirkt die vorherige Belegung von PWD nicht.

Aufgabe 5:

Auch wenn man für das Verschlüsseln von emails eher mit PGP arbeiten würde, nehmen wir doch mal an, Sie wollten eine Datei `brief` mittels `crypt` verschlüsseln und den verschlüsselten Text als datei `xbrief` an einen Teilnehmer `buddy` verschicken. Welche der folgenden Aussagen sind richtig?

(X) Die Verschlüsselung könnte mit `crypt <brief >xbrief` unter Eingabe des Schlüssels (passphrase) erfolgen.

(X) Der Versand würde mit `mail buddy <xbrief` erfolgen.

(X) Der Empfänger `buddy` müßte den Schlüssel kennen, sonst kann er den Inhalt nicht im Klartext lesen.

( ) Eine sehr sichere Methode wäre es, den Schlüssel in die Betreff-Zeile zu schreiben.

( ) Geht nicht, da das `mail`-Kommando keine verschlüsselten Texte versenden kann.

Aufgabe 6:

In unserer Sammlung gefährlicher und dummer Dateinamen betrachten wir heute den leeren Namen, der aus null Zeichen besteht. Kann man ihn mit `touch` erzeugen?

( ) Ja, kann man z.B. mit `touch \` erzeugen.

( ) Ja, kann man z.B. mit `touch ""` erzeugen, aber ohne Leerzeichen dazwischen.

( ) Ja, kann man sogar ganz locker mit `touch` erzeugen.

( ) Ja, kann man mit `touch ''` erzeugen, aber ohne Leerzeichen dazwischen.

( ) Ja, kann man mit `touch ‘echo -n‘` erzeugen.

(X) Nein, leere Dateinamen gibt es nicht.

Aufgabe 7:

Wir betrachten die drei Kommandos (a) - (c). Welche der Ausgaben sollten immer gleich sein, unabhängig vom aktuellen Inhalt von `klausur`?

Hinweis: `ls -ia` liefert alle Dateinamen (auch versteckte) mit vorangestelltem i-node.

(a)
```
wegner@rapunzel:~/klausur$ ls -ia | cat -
...
```

(b)
```
wegner@rapunzel:~/klausur$ ls -ia | sort
...
```

(c)
```
wegner@rapunzel:~/klausur$ ls -ia | sort -k 2
...
```

Die Ausgaben von _____ **(a) und (c)** _____ sind gleich.

Aufgabe 8:

Ähnlich zum Shell-Skript `cx` aus dem Kurs schreiben wir ein Shell-Skript `xweg` mit der einzigen Zeile

```
chmod -x $*
```

und machen `xweg` ausführbar. Danach rufen wir aber `xweg xweg` auf. Was ist richtig?

( ) Das geht gar nicht, weil wir ja im Skript `xweg` uns selbst das Ausführungsrecht entziehen.

(X) Das geht einmal, aber danach kann man `xweg` nicht mehr als ausführbares Shell-Skript aufrufen.

( ) Das geht gar nicht, weil im Skript `chmod -x *` stehen müßte, nicht `$*`.

( ) Das geht nicht, weil die Shell erkennt, dass das Argument und der Kommandoname identisch sind, was verboten ist.

Aufgabe 9:

Der grundsätzliche Vorteil eines Softlinks (eines weichen Verweises) gegenüber dem Hardlink (harten Verweis) ist, dass der Softlink

(X) über Partitionen (volumes) hinweg funktioniert.

( ) prüft, ob die Zieldatei auch existiert.

( ) den Linkzähler in jedem Fall richtig hochsetzt.

( ) den Rechner sehr viel weniger belastet als ein harter Verweis.

( ) `root` als Besitzer zuläßt, was beim Hardlink aus Sicherheitsgründen verboten ist.

Aufgabe 10:

Ergänzen Sie die fehlende Zeile unten!

```
wegner@rapunzel:~/bin$ mkdir leerdir
wegner@rapunzel:~/bin$ ls leerdir | wc
      0       0       0
wegner@rapunzel:~/bin$ ls leerdir -a | wc

      __ 2 _____ 2 _____ 5 _____
```

Aufgabe 11:

Unter einem Zombie-Prozess, versteht man in UNIX einen Prozess, ...

( ) der zum Ende gekommen ist, aber dessen Vaterprozess nicht mehr existiert.

(X) der zum Ende gekommen ist, dessen Vaterprozess aber noch kein `wait` gemacht hat.

( ) der durch `fork` ohne folgendes `exec` entstanden ist.

( ) der ohne `fork` durch Überlagerung (`exec`) der aktuellen Shell entstand.

( ) den man `root` mittels `chown` übertragen wollte, was aber inzwischen verboten ist.

Aufgabe 12:

In einer Multiuser-Umgebung mit gemeinsam genutztem Drucker ist ein häufiges Problem, dass der Drucker blockiert ist, weil der zugehörige Dämonprozess abgestürzt ist. Wie heißt der Prozess für den Standarddrucker `/dev/lp0`, welche Aufgabe hat er und in welcher Beziehung steht er zu `lpr`? Es reichen 3 - 5 Sätze Antwort auf der Rückseite.

Der Dämonprozess heißt lpd und wird durch lpr beauftragt, der direkte Zugriff auf lp0 ist Normalbenutzern verboten. Er sichert die Trennung der Druckerjobs im Mehrbenutzerbetrieb.

Aufgabe 13:

Von Zeit zu Zeit wird der Sysadmin einen UNIX-Server herunterfahren. Was ist richtig?

( ) „Stecker ziehen“ im laufenden Betrieb ist die Regelmethode, um UNIX-Systeme herunterzufahren.

(X) UNIX-Systeme werden in der Regel mit dem `shutdown`-Befehl heruntergefahren.

( ) UNIX-Systeme werden in der Regel durch `kill -9 pager` heruntergefahren.

(X) Zum Auslösen des `shutdown`-Befehls braucht man `root`-Berechtigung.

( ) Solange mindestens ein Benutzer außer `root` angemeldet ist, läßt sich der Rechner nicht (gegen dessen Willen) herunterfahren.

Aufgabe 14:

Ergänzen Sie die Lücken im folgenden Shell-Skript `mydelete`, das alle Normaldateien oder **leere** Verzeichnisse löscht, die als Argumente übergebenen werden. Natürlich nur, sofern die Rechte dies erlauben.

Hinweis: `test -n` *string* liefert wahr genau dann, wenn die Länge von *string* größer Null ist.

```
#!/bin/bash
while [ -n "____ $* ____" ]; do
    if [ -d "_____ $1 _____" ]; then
        __ rmdir __ "$1";
    else
        __rm __ "$1";
    fi
    __ shift __
done
```

Aufgabe 15:

Der folgende Programmteil soll Aufgabe 14 dahingehend ergänzen, dass bei **nichtleeren** Verzeichnissen gefragt wird, ob das Verzeichnis mit allen Inhalten trotzdem gelöscht werden soll. Wenn die eingelesene Antwort „y“ oder „Y“ ist, dann löschen Sie das Verzeichnis rekursiv.

Hinweis: `read -n` *nchars var* liest nur *nchars* Zeichen, weist sie der Variablen *var* zu und kehrt sofort zurück, ohne auf eine komplette Eingabezeile zu warten.

```
# an richtiger Stelle einsetzen bzw. ersetzen
    if [ __ ‘ __ls "$1"|wc -l___‘ ___ -eq 0 ]; then
        #directory empty
        ___ rmdir __ "$1";
    else
        echo "Directory not empty. Delete anyway?"
        # read 1 key
        read -n 1 __ notempty __
        # clear line
        echo
        if [ "$notempty" = y ] _ || __ [ "$notempty" = Y ];
then
            __ rm -r ___ "$1";
        fi
    fi
# Ende Einsetzung Aufgabe 15
```

ENDE DER KLAUSUR

Prof. Dr. Lutz Wegner
Dipl.-Math. Dipl.-Inf. Sebastian Pape **31. Juli 2009**

Einführung in UNIX
Klausur zum Sommersemester 2009

Name:.....Vorname:.....

Matr. Nr.:.....Studiengang:.....

Bearbeiten Sie alle Fragen! Bei Ankreuzaufgaben kann mehr als eine Antwort richtig sein. Hilfsmittel sind nicht zugelassen! Falls Sie für die Beantwortung der Aufgaben zusätzlichen Platz benötigen, verwenden Sie die Rückseiten.

Aufgabe	Punkte max.	Punkte erreicht
1	3	
2	2	
3	2	
4	2	
5	4	
6	3	
7	4	
8	2	
9	2	
10	2	
11	3	
12	2	
13	2	
14	2	
15	2	
16	3	
Summe	**40**	

Aufgabe 1

Nennen Sie drei verbreitete Shells unter Unix. Hinweis: Die im Skript genannte „Visual Shell“ ist nicht mehr aktuell.

Aufgabe 2

Nennen Sie ein Kommando, mit dem man herausfindet, wer sonst noch angemeldet ist!

Aufgabe 3

Kopieren Sie die Passwortdatei in Ihr Heimatverzeichnis unter dem selben Namen.

Aufgabe 4

Benennen Sie diese Kopie der Passwort-Datei um in `passwd.old`!

Aufgabe 5

Erläutern Sie die den folgenden Kommandoablauf auf der Rückseite! Erklären Sie die Ausgabe von `diff`!

```
wegner@rapunzel:~$ ps
  PID TTY          TIME CMD
 5757 pts/0    00:00:00 bash
 5778 pts/0    00:00:00 ps
wegner@rapunzel:~$ ps >psfile1
wegner@rapunzel:~$ ps | cat >psfile2
wegner@rapunzel:~$ diff psfile*
3c3,4
<  5779 pts/0    00:00:00 ps
---
>  5780 pts/0    00:00:00 ps
>  5781 pts/0    00:00:00 cat
```

Aufgabe 6

Schreiben Sie ein Einzeiler-Shellskript `meincopy`, das eine Datei *f* als erstes Argument nimmt und eine Kopie davon im als zweites Argument genannten Verzeichnis *d* unter dem Namen `Kopie-von-`*f* ablegt.
Beispiel: `meincopy psfile2 .` erzeugt eine Datei `Kopie-von-psfile2` im gegenwärtigen Katalog.

Aufgabe 7

Schreiben Sie ein Shellskript `meinsave`, das **alle** Dateien in einem Verzeichnis *d*, das als erstes Argument angegeben wird, in ein dort (also in *d*) anzulegendes Unterverzeichnis mit dem Namen `Kopie-von-`*d* mittels des oben genannten Shellskripts `meincopy` kopiert.
Hinweis: `test -f` prüft, ob das Argument eine Normaldatei ist.

```
#!/bin/bash

mkdir "$1/____________________"

for i in ____________________; do

    if ___________________________________; then

    meincopy "$i" "_____________________________"

    fi

done
```

Aufgabe 8

Kommentieren Sie die folgende Aussage: Damit an einem Shell-Skript das SUID-Bit gesetzt werden kann, muss auch das Schreibrecht an dem Programm für alle gesetzt werden. Bei Programmen mit SUID-Bit, die als binary vorliegen, kann man das machen, muss es aber nicht unbedingt.

Aufgabe 9

Ruft man `ls -l | sort` für ein Verzeichnis mit Unterverzeichnissen, Links und Normaldateien auf, erscheinen in der Ausgabe zuerst alle Verzeichnisse, dann unter Linux die Zeile `insgesamt` ..., dann Dateien, die einen Link darstellen, dann Normaldateien. Begründen Sie diese Reihenfolge auf der Rückseite!

Aufgabe 10

Wie lautet der alternative Kommandoname für das Kommando Punkt „`.`“?

( ) `sh`

( ) `source`

( ) `exec`

( ) `bin`

( ) keines davon, sondern _____.

Aufgabe 11

Ergänzen Sie die fehlende Ausgabe des letzten `ls`-Kommandos.

```
wegner@rapunzel:~/kurs$ mkdir leer
wegner@rapunzel:~/kurs$ cd leer
wegner@rapunzel:~/kurs/leer$ ls | wc
      0       0       0
wegner@rapunzel:~/kurs/leer$ ls -a | wc
_____________________________________
```

Aufgabe 12

In unserer Sammlung gefährlicher und dummer Dateinamen betrachten wir heute den Namen, der nur aus dem Stern „*“ besteht. Kann man ihn erzeugen?

( ) Ja, kann man z. B. mit `touch \*` erzeugen.

( ) Ja, kann man z. B. mit `touch "*"` erzeugen.

( ) Ja, kann man sogar ganz locker mit `touch *` erzeugen, allerdings nur, wenn es nicht schon andere Dateinamen im Verzeichnis gibt.

( ) Ja, kann man sogar ganz locker mit `touch *` erzeugen, auch dann, wenn es schon andere Dateinamen im Verzeichnis gibt.

( ) Nein, `*` ist ein Shell-Metazeichen (Joker) und damit verboten.

Aufgabe 13

```
wegner@rapunzel:~/kurs/leer$ ln -s a b
wegner@rapunzel:~/kurs/leer$ ln -s b a
wegner@rapunzel:~/kurs/leer$ echo Hallo Leute >a
-bash: a: Zu viele Ebenen aus symbolischen Links
```

Obwohl wir nur zwei Links angelegt haben, weigert sich die Bash den Text `Hallo Leute` in die Datei `a` zu schreiben. Begründung!

Aufgabe 14

Die i-Nummer einer Datei *d* wird wo gespeichert?

( ) Im Verzeichnis zusammen mit dem Dateinamen *d*.

( ) Im Vaterverzeichnis des Verzeichnisses, in dem *d* steht.

( ) Im i-Knoten (inode) der Datei *d*.

( ) In den ersten zwei Bytes des Datenbereichs (des 1. Datenblocks) der Datei *d*.

( ) In der `ar`-Tabelle (access rights table) in `/dev`.

( ) Hängt davon ab, wie die Datei erzeugt wurde, z.B. ob die Datei durch einen hard-link erzeugt wurde.

Aufgabe 15

Welche der vier folgenden Kommandozeilen liefert die einzelne Ausgabezeile `/usr/bin/which`?

( ) `man which`

( ) `which man`

( ) `which which`

( ) `which`

( ) `whatis which`

Aufgabe 16

Das `cal`-Kommando unter Linux liefert schöne Kalenderausgaben, etwa diese hier.

```
wegner@rapunzel:~/kurs/leer$ cal 7 2009
     Juli 2009
So Mo Di Mi Do Fr Sa
          1  2  3  4
 5  6  7  8  9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
26 27 28 29 30 31

wegner@rapunzel:~/kurs/leer$ cal 7 2009 | wc
        8      40     168
wegner@rapunzel:~/kurs/leer$ cal 2 2009 | wc -w
```

_____

Ergänzen Sie die fehlende Ausgabe von **`wc -w`** in der letzten Zeile? Das Jahr 2009 ist kein Schaltjahr.

ENDE DER KLAUSUR

Prof. Dr. Lutz Wegner
Dipl.-Math. Dipl.-Inf. Sebastian Pape **31. Juli 2009**

Einführung in UNIX
Klausur zum Sommersemester 2009

Name:..... **Vorname:.....**

Matr. Nr.:..... **Studiengang:.....**

MUSTERLÖSUNG

Bearbeiten Sie alle Fragen! Bei Ankreuzaufgaben kann mehr als eine Antwort richtig sein. Hilfsmittel sind nicht zugelassen! Falls Sie für die Beantwortung der Aufgaben zusätzlichen Platz benötigen, verwenden Sie die Rückseiten.

Aufgabe	Punkte max.	Punkte erreicht
1	3	
2	2	
3	2	
4	2	
5	4	
6	3	
7	4	
8	2	
9	2	
10	2	
11	3	
12	2	
13	2	
14	2	
15	2	
16	3	
Summe	**40**	

Aufgabe 1

Nennen Sie drei verbreitete Shells unter Unix. Hinweis: Die im Skript genannte „Visual Shell“ ist nicht mehr aktuell.

z. B. bash, sh (Bourne-Shell), ksh (Korn-Shell), csh (C-Shell)

Aufgabe 2

Nennen Sie ein Kommando, mit dem man herausfindet, wer sonst noch angemeldet ist!

who (auch w, last)

Aufgabe 3

Kopieren Sie die Passwortdatei in Ihr Heimatverzeichnis unter dem selben Namen.

cp /etc/passwd ~

Aufgabe 4

Benennen Sie diese Kopie der Passwort-Datei um in `passwd.old`!

mv ~/passwd ~/passwd.old

Aufgabe 5

Erläutern Sie die den folgenden Kommandoablauf auf der Rückseite! Erklären Sie die Ausgabe von `diff`!

```
wegner@rapunzel:~$ ps
  PID TTY          TIME CMD
 5757 pts/0    00:00:00 bash
 5778 pts/0    00:00:00 ps
wegner@rapunzel:~$ ps >psfile1
wegner@rapunzel:~$ ps | cat >psfile2
wegner@rapunzel:~$ diff psfile*
3c3,4
<  5779 pts/0    00:00:00 ps
---
>  5780 pts/0    00:00:00 ps
>  5781 pts/0    00:00:00 cat
```

Es werden zwei Ausgaben von ps in zwei Dateien abgelegt, beim ersten Aufruf durch direkte Umleitung, beim zweiten durch eine Pipe und cat. Die Dateiinhalte werden dann mit diff verglichen. Die Inhalte der Dateien unterscheiden sich einmal wegen der unterschiedlichen Prozessnummern bei ps, dann noch wegen des zusätzlichen cat-Kommandos beim zweiten Aufruf.

Aufgabe 6

Schreiben Sie ein Einzeiler-Shellskript `meincopy`, das eine Datei *f* als erstes Argument nimmt und eine Kopie davon im als zweites Argument genannten Verzeichnis *d* unter dem Namen `Kopie-von-`*f* ablegt.
Beispiel: `meincopy psfile2 .` erzeugt eine Datei `Kopie-von-psfile2` im gegenwärtigen Katalog.

```
#!/bin/bash                                  ← auch richtig ohne diese Angabe
/bin/cp "$1" "$2"/Kopie-von-"$1"             ← auch richtig mit nur cp
```

Aufgabe 7

Schreiben Sie ein Shellskript `meinsave`, das **alle** Dateien in einem Verzeichnis *d*, das als erstes Argument angegeben wird, in ein dort (also in *d*) anzulegendes Unterverzeichnis mit dem Namen `Kopie-von-`*d* mittels des oben genannten Shellskripts `meincopy` kopiert.
Hinweis: `test -f` prüft, ob das Argument eine Normaldatei ist.

```
#!/bin/bash

mkdir "$1/Kopie-von$1"

for i in `ls -a "$1"`; do

    if [ -f "$i" ]; then

    meincopy "$i" "$1/Kopie-von-$1"

    fi

done
```

Aufgabe 8

Kommentieren Sie die folgende Aussage: Damit an einem Shell-Skript das SUID-Bit gesetzt werden kann, muss auch das Schreibrecht an dem Programm für alle gesetzt werden. Bei Programmen mit SUID-Bit, die als binary vorliegen, kann man das machen, muss es aber nicht unbedingt.

Die erste Aussage ist falsch, das SUID-Bit läßt sich zwar setzen (unabhängig von anderen Bits), hat aber bei einem Shell-Skript keine Wirkung. Die zweite Aussage ist insofern unsinnig, als ein Schreibrecht bei einem Programm mit SUID-Bit aus Sicherheitsgründen keine gute Idee ist.

Aufgabe 9

Ruft man `ls -l | sort` für ein Verzeichnis mit Unterverzeichnissen, Links und Normaldateien auf, erscheinen in der Ausgabe zuerst alle Verzeichnisse, dann unter Linux die Zeile `insgesamt ...`, dann Dateien, die einen Link darstellen, dann Normaldateien. Begründen Sie diese Reihenfolge auf der Rückseite!

Die Reihenfolge der Ausgabe wird durch das erste Zeichen jeder Zeile bei der Auflistung mit ls -l bestimmt. Dies ist bei Verzeichnissen ein "d", das kommt vor dem Text „insgesamt ...", dem "l" für Links und zuletzt dem "-" bei Normaldateien.

Aufgabe 10

Wie lautet der alternative Kommandoname für das Kommando Punkt „.“?

( ) `sh`

(X) `source`

( ) `exec`

( ) `bin`

( ) keines davon, sondern _____.

Aufgabe 11

Ergänzen Sie die fehlende Ausgabe des letzten `ls`-Kommandos.

```
wegner@rapunzel:~/kurs$ mkdir leer
wegner@rapunzel:~/kurs$ cd leer
wegner@rapunzel:~/kurs/leer$ ls | wc
      0       0       0
wegner@rapunzel:~/kurs/leer$ ls -a | wc
______2______2______5__________________
```

Aufgabe 12

In unserer Sammlung gefährlicher und dummer Dateinamen betrachten wir heute den Namen, der nur aus dem Stern „*“ besteht. Kann man ihn erzeugen?

(X) Ja, kann man z. B. mit `touch \*` erzeugen.

(X) Ja, kann man z. B. mit `touch "*"` erzeugen.

(X) Ja, kann man sogar ganz locker mit `touch *` erzeugen, allerdings nur, wenn es nicht schon andere Dateinamen im Verzeichnis gibt.

( ) Ja, kann man sogar ganz locker mit `touch *` erzeugen, auch dann, wenn es schon andere Dateinamen im Verzeichnis gibt.

( ) Nein, `*` ist ein Shell-Metazeichen (Joker) und damit verboten.

Aufgabe 13

```
wegner@rapunzel:~/kurs/leer$ ln -s a b
wegner@rapunzel:~/kurs/leer$ ln -s b a
wegner@rapunzel:~/kurs/leer$ echo Hallo Leute >a
-bash: a: Zu viele Ebenen aus symbolischen Links
```

Obwohl wir nur zwei Links angelegt haben, weigert sich die Bash den Text `Hallo Leute` in die Datei `a` zu schreiben. Begründung!

Es werden zirkuläre symbolische Links angelegt, a zeigt auf b und b wiederum auf a.

Aufgabe 14

Die i-Nummer einer Datei *d* wird wo gespeichert?

(X) Im Verzeichnis zusammen mit dem Dateinamen *d*.

( ) Im Vaterverzeichnis des Verzeichnisses, in dem *d* steht.

( ) Im i-Knoten (inode) der Datei *d*.

( ) In den ersten zwei Bytes des Datenbereichs (des 1. Datenblocks) der Datei *d*.

( ) In der `ar`-Tabelle (access rights table) in `/dev`.

( ) Hängt davon ab, wie die Datei erzeugt wurde, z.B. ob die Datei durch einen hard-link erzeugt wurde.

Aufgabe 15

Welche der vier folgenden Kommandozeilen liefert die einzelne Ausgabezeile `/usr/bin/which`?

( ) `man which`

( ) `which man`

(X) `which which`

( ) `which`

( ) `whatis which`

Aufgabe 16

Das `cal`-Kommando unter Linux liefert schöne Kalenderausgaben, etwa diese hier.

```
wegner@rapunzel:~/kurs/leer$ cal 7 2009
     Juli 2009
So Mo Di Mi Do Fr Sa
          1  2  3  4
 5  6  7  8  9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
26 27 28 29 30 31

wegner@rapunzel:~/kurs/leer$ cal 7 2009 | wc
        8      40     168
wegner@rapunzel:~/kurs/leer$ cal 2 2009 | wc -w

        _____37_____
```

Ergänzen Sie die fehlende Ausgabe von **`wc -w`** in der letzten Zeile? Das Jahr 2009 ist kein Schaltjahr.

ENDE DER KLAUSUR

Prof. Dr. Lutz Wegner
Dipl.-Math. Dipl.-Inf. Sebastian Pape **15. Februar 2009**

Einführung in UNIX
Klausur zum Wintersemester 2009/10

Name:.....Vorname:.....

Matr. Nr.:.....Studiengang:.....

Bearbeiten Sie alle Fragen! Bei Ankreuzaufgaben kann mehr als eine Antwort richtig sein. Hilfsmittel sind nicht zugelassen! Falls Sie für die Beantwortung der Aufgaben zusätzlichen Platz benötigen, verwenden Sie die Rückseiten.

Aufgabe	Punkte max.	Punkte erreicht
1	2	
2	4	
3	2	
4	6	
5	4	
6	4	
7	4	
8	2	
9	2	
10	2	
11	2	
12	2	
13	2	
14	4	
15	2	
16	2	
17	4	
Summe	**50**	

Aufgabe 1

In welchem Verzeichnis sind heute üblicherweise in Linux - anders als im Skript dargestellt - die Startverzeichnisse der Teilnehmer eingetragen?

Aufgabe 2

Geben Sie zwei alternative Kommandozeilen an, mit denen eine leere Datei `Nullo`, die bisher nicht existiert, anlegt würde!

Aufgabe 3

Auch wenn heute Faschingsmontag ist, verstehen wir nicht, warum bei `cp` das Verzeichnis `leerdir` fröhlich sein soll. Erläuterung!

```
mkdir leerdir
cp leerdir leerdir2
cp: Verzeichnis »leerdir« ausgelassen
```

Aufgabe 4

In einem bekannten, unausprechlichen Betriebssystem kann man markierte Dateien durch Mausklick kopieren, die dann im gleichen Verzeichnis unter dem Dateinamen `Kopie von` *Name* abgelegt werden. Was macht das folgende Shellskript `meincopy` und in welchen zwei offensichtlichen Fällen funktioniert es nicht?

```
for name in $*
do
    cp "$name" "Kopie von $name"
done
```

Aufgabe 5

Erläutern Sie den folgenden Kommandoablauf auf der Rückseite! Warum listet das `ps`-Kommando `echo` nicht auf? Wieso wurden die NL-Zeichen durch Leerzeichen ersetzt?

```
fix@labserv:~$ ps
PID TTY          TIME CMD
17710 pts/0    00:00:00 bash
17726 pts/0    00:00:00 ps
fix@labserv:~$ echo `ps`
PID TTY TIME CMD 17710 pts/0 00:00:00 bash 17727 pts/0 00:00:00 ps
fix@labserv:~$
```

Aufgabe 6

Das built-in (in der Shell ablaufende) `time`-Kommando misst bekanntlich die Laufzeit des folgenden Kommandos. Erläutern Sie auf der Rückseite den Ablauf unten, speziell auch, warum `f2` beim 2. Versuch leer ist.

```
fix@labserv:~$ (time date >f1) 2>f2
fix@labserv:~$ cat f2

real    0m0.001s
user    0m0.000s
sys     0m0.000s
fix@labserv:~$ time date >f1 2>f2

real    0m0.034s
user    0m0.000s
sys     0m0.004s
fix@labserv:~$ cat f2
fix@labserv:~$
```

Aufgabe 7

Das `who`-Kommando mit der Option `-q` liefert die Login-Namen der angemeldeten Teilnehmer und deren Anzahl. Erläutern Sie kurz was jedes der Kommandos in der Pipe bewirkt!

```
fix@labserv:~$ who -q
fix fix pape s_6aonf3 s_f38s5k pape
# Benutzer=6
fix@labserv:~$ who -q | head -1 | tr ' ' '\n' | sort | uniq
fix
pape
s_6aonf3
s_f38s5k
```

Aufgabe 8

Wenn ein Prozess sich beendet, z. B. mit `exit()`, der Vaterprozess aber noch kein `wait()` gemacht hat, dann wird der beendete Prozess zum

( ) Zombie-Prozess.

( ) Waisenkind.

( ) S-Prozess (sleeping)

( ) Hintergrundprozess

( ) Null-Prozess

Aufgabe 9

Das Setzen des SUID-Bits bewirkt bei der Ausführung von Shellskripten nichts. Womit hängt das zusammen?

( ) Man kann an einem Shellskript sowieso kein x-Recht (Ausführungsrecht) setzen.

( ) Der Superuser `root` darf nie Besitzer einer Shell werden.

( ) Das SUID-Bit wird als Nebeneffekt eines `exec`-Aufrufs gesetzt, was nur für binaries geht.

( ) Im i-node der Datei, die das Skript abspeichert, ist kein Platz für das SUID-Bit vorgesehen.

( ) Der Superuser ist sowieso Besitzer aller Shellskripte und leiht auch seine Rechte dem Aufrufer (weil ihm auch die Shell gehört).

Aufgabe 10

Nehmen wir an, `prog1` sei ein ausführbares Programm und `prog2` sei ein per

```
ln prog1 prog2
```

vergebener Alias-Name für das Programm. Was ist richtig?

( ) Ruft man `prog2` auf, während noch `prog1` aktiv ist, teilen sich beide den Prozessidentifier.

( ) `prog1` und `prog2` haben die selbe i-Nummer (erkennbar bei ls -i)

( ) `prog2` ist zwar ein Alias-Name für die Datei, aufrufen kann man das Programm aber nur als `prog1`.

( ) Diese Form der Verknüpfung geht nur per Softlink (`ln -s`).

( ) Das Ausführungsrecht kann nie bei zwei mit `ln` verknüpften Dateien gleichzeitig gesetzt sein (ehemalige Sicherheitslücke).

Aufgabe 11

Welches der folgenden Kommandos ist kein builtin Kommando, das die `bash` selbst ausführt?

( ) `:`

( ) `source`

( ) `cd`

( ) `man`

( ) `read`

( ) `write`

Aufgabe 12

Welches der folgenden Kommandos liefert den Namen des Arbeitsverzeichnisses?

( ) `ls -d` `` `pwd` ``

( ) `ls -d`

( ) `ls -d .`

( ) `ls -d ~`

( ) `ls -d $PWD`

Aufgabe 13

In unserer Sammlung gefährlicher und dummer Dateinamen betrachten wir heute den Namen, der nur aus dem Escapezeichen „\“ besteht. Kann man ihn erzeugen?

( ) Ja, kann man z. B. mit `touch \\` erzeugen.

( ) Ja, kann man z. B. mit `touch "\"` erzeugen.

( ) Ja, kann man sogar ganz locker mit `touch \` erzeugen.

( ) Ja, kann man mit `touch '\'` erzeugen.

( ) Nein, `\` ist das Shell-Fluchtsymbol und damit als Dateiname verboten.

Aufgabe 14

Kommentieren Sie den folgenden Kommandoaufruf und die Reaktion der Shell. Warum ist die Reaktion der Shell sinnvoll?

```
fix@labserv:~/aushang$  UID=0 bash
-bash: UID: readonly variable
fix@labserv:~/aushang$
```

Aufgabe 15

Wird ein Programm, z. B. die Shell oder ein Browser, mehrfach aufgerufen von unterschiedlichen Nutzern, dann kann Unix die Effizienz steigern indem

( ) die Inkarnationen (Prozesse) sich den Prozessidentifier teilen.

( ) die Inkarnationen (Prozesse) sich das Textsegment teilen, der Code also nur einmal angelegt (geladen) wird.

( ) die Inkarnationen (Prozesse) sich die Dateideskriptoren teilen.

( ) die Inkarnationen (Prozesse) sich das Stacksegment teilen, es also nur einmal angelegt wird.

( ) neue (weitere) Inkarnationen (Prozesse) sich per `fork` ohne `exec` erzeugen lassen.

Aufgabe 16

Nehmen wir an, Sie hätten vor, in nächster Zeit brisante Daten auf eine CD zu brennen. Wo würden Sie den Eintrag

```
brw-rw---- 1 root cdrom 11, 0  4. Feb 07:16 /???/sr0
```

finden? Ersetzen Sie die drei Fragezeichen.

Aufgabe 17

Ergänzen Sie die Lücken im folgenden Shell-Skript, das nacheinander für jedes Argument prüft, ob der Name ein Verzeichnis ist und wenn ja, die Anzahl seiner Einträge ausgibt.

```
for _____________________________
do
    if [ -d "$name" ]
    then
        echo `ls -a "$name" | _________________` \
            Eintraege im Verzeichnis "$name"
    else
    echo Fehler: "$name" kein Verzeichnis
    fi
done
```

ENDE DER KLAUSUR

Prof. Dr. Lutz Wegner
Dipl.-Math. Dipl.-Inf. Sebastian Pape **15. Februar 2009**

Einführung in UNIX
Klausur zum Wintersemester 2009/10

MUSTERLÖSUNG

Name:.....Vorname:.....

Matr. Nr.:.....Studiengang:.....

Bearbeiten Sie alle Fragen! Bei Ankreuzaufgaben kann mehr als eine Antwort richtig sein. Hilfsmittel sind nicht zugelassen! Falls Sie für die Beantwortung der Aufgaben zusätzlichen Platz benötigen, verwenden Sie die Rückseiten.

Aufgabe	Punkte max.	Punkte erreicht
1	2	
2	4	
3	2	
4	6	
5	4	
6	4	
7	4	
8	2	
9	2	
10	2	
11	2	
12	2	
13	2	
14	4	
15	2	
16	2	
17	4	
Summe	**50**	

Aufgabe 1

In welchem Verzeichnis sind heute üblicherweise in Linux - anders als im Skript dargestellt - die Startverzeichnisse der Teilnehmer eingetragen?

/home

Aufgabe 2

Geben Sie zwei alternative Kommandozeilen an, mit denen eine leere Datei `Nullo`, die bisher nicht existiert, anlegt würde!

touch Nullo
>Nullo, oder auch echo -n >Nullo

Aufgabe 3

Auch wenn heute Faschingsmontag ist, verstehen wir nicht, warum bei `cp` das Verzeichnis `leerdir` fröhlich sein soll. Erläuterung!

```
mkdir leerdir
cp leerdir leerdir2
cp: Verzeichnis »leerdir« ausgelassen
```

cp kopiert (ohne die Option -r) keine Verzeichnisse.

Aufgabe 4

In einem bekannten, unausprechlichen Betriebssystem kann man markierte Dateien durch Mausklick kopieren, die dann im gleichen Verzeichnis unter dem Dateinamen `Kopie von` *Name* abgelegt werden. Was macht das folgende Shellskript `meincopy` und in welchen zwei offensichtlichen Fällen funktioniert es nicht?

```
for name in $*
do
    cp "$name" "Kopie von $name"
done
```

Von allen als Argument übergebenen Dateien wird eine Kopie mit dem neuen Namen *Kopie von ...* im Arbeitsverzeichnis abgelegt. Funktioniert nicht bei vollen Pfadnamen und bei Verzeichnissen oder wenn die Datei nicht existiert oder der Benutzer kein Schreibrecht auf dem Verzeichnis hat.

Die Aussage „funktioniert nicht, wenn Kopie von ... schon existiert" ist falsch, denn es wird trotzdem kopiert. Die Aussage „Funktioniert nicht, wenn keine Argumente übergeben werden" ist eigentlich auch falsch, da das Skript ohne Fehlerausgabe durchläuft.

Aufgabe 5

Erläutern Sie den folgenden Kommandoablauf auf der Rückseite! Warum listet das `ps`-Kommando `echo` nicht auf? Wieso wurden die NL-Zeichen durch Leerzeichen ersetzt?

```
fix@labserv:~$ ps
PID TTY          TIME CMD
17710 pts/0    00:00:00 bash
17726 pts/0    00:00:00 ps
fix@labserv:~$ echo `ps`
PID TTY TIME CMD 17710 pts/0 00:00:00 bash 17727 pts/0 00:00:00 ps
fix@labserv:~$
```

echo ist builtin-Kommando, daher keine Auflistung.
NL-Ersetzung wegen Kommandosubstitution

Aufgabe 6

Das built-in (in der Shell ablaufende) `time`-Kommando misst bekanntlich die Laufzeit des folgenden Kommandos. Erläutern Sie auf der Rückseite den Ablauf unten, speziell auch, warum `f2` beim 2. Versuch leer ist.

```
fix@labserv:~$ (time date >f1) 2>f2
fix@labserv:~$ cat f2

real    0m0.001s
user    0m0.000s
sys     0m0.000s
fix@labserv:~$ time date >f1 2>f2

real    0m0.034s
user    0m0.000s
sys     0m0.004s
fix@labserv:~$ cat f2
fix@labserv:~$
```

Beim ersten Aufruf bezieht sich die Umlenkung der Standardfehlerausgabe, in die time schreibt, auf die ganze Untershell. Die Ausgabe von date steht in f1.

Im zweiten Aufruf ist 2>f2 gebunden an date, das keine Fehlerausgabe produziert. Die Standardfehlerausgabe von time wird nicht umgelenkt und erscheint sofort.

Aufgabe 7

Das `who`-Kommando mit der Option `-q` liefert die Login-Namen der angemeldeten Teilnehmer und deren Anzahl. Erläutern Sie kurz was jedes der Kommandos in der Pipe bewirkt!

```
fix@labserv:~$ who -q
fix fix pape s_6aonf3 s_f38s5k pape
# Benutzer=6
fix@labserv:~$ who -q | head -1 | tr ' ' '\n' | sort | uniq
fix
pape
s_6aonf3
s_f38s5k
```

who -q liefert die Teilnehmer in einer Zeile (und die Zählung). Von dieser Ausgabe nehmen wir nur die erste Zeile, ersetzen darin Leerzeichen durch NL, sortieren diese Zeilen, so dass gleiche Einträge (wenn ein Teilnehmer mehrfach angemeldet ist) untereinander stehen, die bis auf einen in uniq dann gelöscht werden.

Aufgabe 8

Wenn ein Prozess sich beendet, z. B. mit `exit()`, der Vaterprozess aber noch kein `wait()` gemacht hat, dann wird der beendete Prozess zum

(X) Zombie-Prozess.

( ) Waisenkind.

( ) S-Prozess (sleeping)

( ) Hintergrundprozess

( ) Null-Prozess

Aufgabe 9

Das Setzen des SUID-Bits bewirkt bei der Ausführung von Shellskripten nichts. Womit hängt das zusammen?

( ) Man kann an einem Shellskript sowieso kein x-Recht (Ausführungsrecht) setzen.

( ) Der Superuser `root` darf nie Besitzer einer Shell werden.

(X) Das SUID-Bit wird als Nebeneffekt eines `exec`-Aufrufs gesetzt, was nur für binaries geht.

( ) Im i-node der Datei, die das Skript abspeichert, ist kein Platz für das SUID-Bit vorgesehen.

( ) Der Superuser ist sowieso Besitzer aller Shellskripte und leiht auch seine Rechte dem Aufrufer (weil ihm auch die Shell gehört).

Aufgabe 10

Nehmen wir an, `prog1` sei ein ausführbares Programm und `prog2` sei ein per

```
ln prog1 prog2
```

vergebener Alias-Name für das Programm. Was ist richtig?

( ) Ruft man `prog2` auf, während noch `prog1` aktiv ist, teilen sich beide den Prozessidentifier.

(X) `prog1` und `prog2` haben die selbe i-Nummer (erkennbar bei ls -i)

( ) `prog2` ist zwar ein Alias-Name für die Datei, aufrufen kann man das Programm aber nur als `prog1`.

( ) Diese Form der Verknüpfung geht nur per Softlink (`ln -s`).

( ) Das Ausführungsrecht kann nie bei zwei mit `ln` verknüpften Dateien gleichzeitig gesetzt sein (ehemalige Sicherheitslücke).

Aufgabe 11

Welches der folgenden Kommandos ist kein builtin Kommando, das die `bash` selbst ausführt?

( ) `:`

( ) `source`

( ) `cd`

(X) `man`

( ) `read`

(X) `write`

Aufgabe 12

Welches der folgenden Kommandos liefert den Namen des Arbeitsverzeichnisses?

(X) `ls -d` `` `pwd` ``

( ) `ls -d`

( ) `ls -d .`

( ) `ls -d ~`

(X) `ls -d $PWD`

Aufgabe 13

In unserer Sammlung gefährlicher und dummer Dateinamen betrachten wir heute den Namen, der nur aus dem Escapezeichen „\“ besteht. Kann man ihn erzeugen?

(X) Ja, kann man z. B. mit `touch \\` erzeugen.

( ) Ja, kann man z. B. mit `touch "\"` erzeugen.

( ) Ja, kann man sogar ganz locker mit `touch \` erzeugen.

(X) Ja, kann man mit `touch '\'` erzeugen.

( ) Nein, `\` ist das Shell-Fluchtsymbol und damit als Dateiname verboten.

Aufgabe 14

Kommentieren Sie den folgenden Kommandoaufruf und die Reaktion der Shell. Warum ist die Reaktion der Shell sinnvoll?

```
fix@labserv:~/aushang$  UID=0 bash
-bash: UID: readonly variable
fix@labserv:~/aushang$
```

Die Variable UID (user ID), die den Besitzer eines Prozesses angibt, ist als read-only markiert, kann also nicht mit einem neuen Wert überschrieben werden. Das ist wichtig, weil sonst der Aufrufer mit obiger Zeile eine Superuser-Shell bekäme.

Aufgabe 15

Wird ein Programm, z. B. die Shell oder ein Browser, mehrfach aufgerufen von unterschiedlichen Nutzern, dann kann Unix die Effizienz steigern indem

( ) die Inkarnationen (Prozesse) sich den Prozessidentifier teilen.

(X) die Inkarnationen (Prozesse) sich das Textsegment teilen, der Code also nur einmal angelegt (geladen) wird.

( ) die Inkarnationen (Prozesse) sich die Dateideskriptoren teilen.

( ) die Inkarnationen (Prozesse) sich das Stacksegment teilen, es also nur einmal angelegt wird.

( ) neue (weitere) Inkarnationen (Prozesse) sich per `fork` ohne `exec` erzeugen lassen.

Aufgabe 16

Nehmen wir an, Sie hätten vor, in nächster Zeit brisante Daten auf eine CD zu brennen. Wo würden Sie den Eintrag

```
brw-rw---- 1 root cdrom 11, 0  4. Feb 07:16 /dev/sr0
```

finden? Ersetzen Sie die drei Fragezeichen.

Aufgabe 17

Ergänzen Sie die Lücken im folgenden Shell-Skript, das nacheinander für jedes Argument prüft, ob der Name ein Verzeichnis ist und wenn ja, die Anzahl seiner Einträge ausgibt.

```
for ___ name in $* ___
do
    if [ -d "$name" ]
    then
        echo `ls -a "$name" | ___ wc -l ___` \
            Eintraege im Verzeichnis "$name"
    else
    echo Fehler: "$name" kein Verzeichnis
    fi
done
```

ENDE DER KLAUSUR

Prof. Dr. Lutz Wegner
Dipl.-Math. Dipl.-Inf. Sebastian Pape **26. Juli 2010**

Einführung in UNIX
Klausur zum Sommersemester 2010

Name:.....Vorname:.....

Matr. Nr.:.....Studiengang:.....

Bearbeiten Sie alle Fragen! Bei Ankreuzaufgaben kann mehr als eine Antwort richtig sein. Hilfsmittel sind nicht zugelassen! Falls Sie für die Beantwortung der Aufgaben zusätzlichen Platz benötigen, verwenden Sie die Rückseiten.

Aufgabe	Punkte max.	Punkte erreicht
1	2	
2	2+2+2+2+2	
3	2	
4	4+2+2	
5	6	
6	2	
7	2	
8	2	
9	2	
10	2	
11	2	
12	2	
13	2	
14	2	
15	2	
16	2+2	
Summe	**52**	

Aufgabe 1

Für klassische Unix-Verhältnisse zeigt Linux ein ungewohnt geschwätziges, selbstzweiflerisches Verhalten bei der unten gezeigten Kommandozeile. Erläutern Sie den Sachverhalt.

```
fix@labserv:~$ mail fix </dev/null
No message, no subject; hope that's ok
fix@labserv:~$
```

Aufgabe 2

Kommentieren Sie das Verhalten der Shell im Umgang mit Links auf Verzeichnisse unter Linux.

```
fix@labserv:~$ ln leerdir linkdir
ln: »leerdir«: harte Verknüpfung für Verzeichnisse nicht
erlaubt
fix@labserv:~$
```

(a) Hätte das laut Vorlesung und Skript funktionieren sollen?

```
fix@labserv:~$ ln -s leerdir linkdir
fix@labserv:~$ ls -l
insgesamt 84
...
drwxr-xr-x 2 fix fix  4096  8. Feb 15:32 leerdir
lrwxrwxrwx 1 fix fix     7  9. Jul 16:12 linkdir -> leerdir
...
fix@labserv:~$
```

(b) Was ist `linkdir` und warum ist die Dateigröße von `linkdir` 7 Bytes?

```
fix@labserv:~$ ls -l linkdir
lrwxrwxrwx 1 fix fix 7  9. Jul 16:12 linkdir -> leerdir
fix@labserv:~$
```

(c) Was zeigt `ls -l` hier? Was hätte man sich vielleicht gewünscht?

```
fix@labserv:~$ echo hallo >linkdir/hallofile
fix@labserv:~$ ls -l leerdir
insgesamt 4
-rw-r--r-- 1 fix fix 6  9. Jul 16:13 hallofile
fix@labserv:~$ ls -l linkdir/.
insgesamt 4
-rw-r--r-- 1 fix fix 6  9. Jul 16:13 hallofile
fix@labserv:~$
```

(d) Funktioniert `linkdir` hier wie ein Verzeichnis? Kurze Begründung was die Shell macht!

```
fix@labserv:~$ cd linkdir
fix@labserv:~/linkdir$ ls
hallofile
fix@labserv:~/linkdir$ pwd
/home/fix/linkdir
fix@labserv:~/linkdir$
```

(e) Kann man `linkdir` zum Arbeitsverzeichnis machen?

Aufgabe 3

Bei einigen Teilnehmern ist in `/etc/passwd` an der Stelle für die login-Shell `/bin/false` eingetragen, also keine Shell, z. B. beim Eintrag
`postfix:x:111:119::/var/spool/postfix:/bin/false`
Welche Aussage ist richtig, bzw. welche sind richtig?

( ) Liefert beim versuchten Aufruf sofort einen Fehlerstatuscode ab.

( ) Die eigentliche Shell wird beim Systemstart erst eingetragen.

( ) `/bin/false` ist tatsächlich eine Shell, die sog. Alias-Shell (free alias shell extension)

( ) Der Eintrag ist eigentlich irrelevant, er könnte auch `/usr/bin/vi` sein.

( ) Selbst der Teilnehmer `root` hat dort diesen Eintrag `/bin/false`.

Aufgabe 4

In der Vorlesung und im Skript werden die sog. *compound commands* der Shell mit Ausnahme der Untershell nicht behandelt. So wertet `((`*expression*`))` einen arithmetischen Ausdruck, also *expression* aus. Ist das Ergebnis der Auswertung ungleich Null liefert es einen Statuswert 0 zurück, sonst liefert es 1. Mit dem `let`-Kommando kann man den Wert des Ausdrucks einer Variablen zuweisen. Dies verwenden wir im folgenden Skript `mycalc`.

mycalc

```
if [ $# -lt 1 ]; then
  echo "Keine Argumente - keine Auswertung. So isses."
elif (( $* )); then
  let x="$*"
  echo "_____________________"
else
  echo "______ = 0 oder kein arithmetischer Ausdruck"
fi
```

(a) Füllen Sie die Lücken in `mycalc` unter Berücksichtigung der folgenden Aufrufe.

```
fix@labserv:~/aushang$ ../mycalc
Keine Argumente - keine Auswertung. So isses.
fix@labserv:~/aushang$ ../mycalc 3 + 4
3 + 4 = 7
fix@labserv:~/aushang$ ../mycalc 4 - 4
4 - 4 = 0 oder kein arithmetischer Ausdruck
fix@labserv:~/aushang$ ../mycalc 4 4
../mycalc: line 3: ((: 4 4 : syntax error in expression (error
token is "4 ")
4 4 = 0 oder kein arithmetischer Ausdruck
```

(b) Erläutern Sie die folgende Ausgabe!

```
fix@labserv:~/aushang$ ../mycalc 4 * 4
../mycalc: line 3: ((: 4 nullo tcltk 4 : syntax error in
expression (error token is "nullo tcltk 4 ")
4 nullo tcltk 4 = 0 oder kein arithmetischer Ausdruck
```

(c) Wie muss der Aufruf für eine Multiplikation korrekt lauten?

```
fix@labserv:~/aushang$ ../mycalc 4 ______ 4
4 * 4 = 16
```

Aufgabe 5

Unser Shell-Skript `cleanup`, das zum Entfernen leerer Normaldateien auffordert, liefert bei Aufruf im Arbeitsverzeichnis `aushang` die folgende Ausgabe. Ergänzen Sie die Lücken im Shell-Skript. Dateien, die mit Punkt beginnen, brauchen nicht berücksichtigt zu werden.

```
fix@labserv:~/aushang$ ls -l
insgesamt 8
drwxr-xr-x 2 fix fix 4096 14. Jul 11:27 MeinKatalog
-rw-r--r-- 1 fix fix    0 12. Jul 14:24 nullo
-rw-r--r-- 1 fix fix  100 20. Nov 2009  tcltk
fix@labserv:~/aushang$ ../cleanup
MeinKatalog keine leere Datei
rm: reguläre leere Datei »nullo« entfernen? n
tcltk keine leere Datei
fix@labserv:~/aushang$
```

cleanup

```
for name in ____________________
do
  if [ -f "__________" -a ! -s "_________" ]
  then
    rm ____________________
  else
    echo ___________________________________
  fi
done
```

Aufgabe 6

Im Skript und auch im Kurs wird erwähnt, dass das Ändern des Besitzers (und der Besitzergruppe) einer Datei mit `chown` in den meisten neueren Unix- und Linux-Versionen nur noch dem Super-User vorbehalten ist. Aber wie kann ein normaler Anwender für seine Arbeit weiterhin anderen Gruppenrechte einräumen? Funktioniert das `chgrp`-Kommando weiterhin?

( ) Ja, aber nur der Besitzer einer Datei kann die Gruppe dieser Datei auf jede andere Gruppe ändern, in der er Mitglied ist.

( ) Ja, aber nur der Besitzer einer Datei kann die Gruppe dieser Datei auf jede andere Gruppe ändern, auch wenn er nicht Mitglied ist.

( ) Ja, grundsätzlich kann sogar jeder Anwender die Besitzergruppe einer Datei ändern, auch wenn er nicht deren Besitzer ist.

( ) Nein, Besitzergruppen für Dateien, Verzeichnisse und Geräte gibt es nicht mehr. Es wird nur noch aus Kompatibilitätsgründen eine Gruppe `group` eingetragen, der aber alle angehören.

( ) Nein, das Kommando zum Ändern der Besitzergruppe heißt `newgrp`.

Aufgabe 7

Welche Aussagen zum i-node (bzw. der i-Nummer) im UNIX-Dateisystem sind richtig?

( ) Jede Normaldatei und jedes Verzeichnis hat genau einen i-node.

( ) In den Verzeichnissen sind beim Dateinamen die zugehörige i-Nummer eingetragen (wegen der harten Links ggf. bei mehreren Namen die selbe i-Nummer).

( ) Im i-node sind unter anderem der owner-id (Besitzer-Id) und die für die Datei gesetzten Rechte eingetragen.

( ) Gehört der i-node zu einem Verzeichnis, ist im i-node selbst die i-Nummer des Vaterverzeichnisses und die i-Nummer des eigenen i-node eingetragen (wegen `.` und `..`).

( ) Wächst eine Datei über eine gewisse Größe (derzeit 2 MB bei Linux), muss auch der i-node wachsen, weil er die Adressen **aller** Datenblöcke aufnimmt.

Aufgabe 8

Kann man sich in einem laufenden Shell-Skript die Prozessnummer (process id) des ausführenden Prozesses anzeigen lassen?

( ) Ja, z. B. mit `echo $$`. Man erhält den Prozessidentifier des Shell,

( ) Ja, z. B. mit echo `$PPID`. Auch so erhält man den Prozessidentifier der ausführenden Shell.

( ) Nein, ein Shell-Skript ist kein executable und hat zur Laufzeit keinen Prozessidentifier.

( ) Nein, dazu müßte man zwingend das Skript in einer Unter-Untershell laufen lassen.

( ) Nur wenn das t-Bit gesetzt ist.

Aufgabe 9

Wenn zwei oder mehr Kommandos über eine Pipe verbunden werden, z. B. bei `sort` *xyz* | `uniq | lpd`, dann ...

( ) ... können die hinteren Kommandos erst starten, wenn die vorderen fertig sind.

( ) ... findet nach jedem weitergegebenen Byte (Zeichen) ein Prozessorwechsel zum nächsten Prozess statt (nach dem letzten wieder zurück zum ersten Prozess in der Pipe).

( ) ... darf keines der beteiligten Kommandos ein Shell-Skript sein.

( ) ... ist bei allen beteiligten Kommandos die Standardfehlerausgabe in die Standardausgabe umgeleitet.

( ) ... schreibt die Standardausgabe des Kommandos links vom Pipesymbol in die Standardeingabe des Kommandos rechts davon.

Aufgabe 10

Nach einem `fork()`-Systemaufruf entstehen zwei identische Prozessabbilder. Woran erkennen die beiden Prozesse, wer Vater und wer Sohn ist?

( ) Am Alter. Die Prozesse fragen die registrierte Laufzeit ab (`ERTIME`, elapsed running time), die beim Sohn 0 und beim Vater > 0 ist (wegen des `fork`-Aufrufs).

( ) Am Befehlszeiger. Der Vater steht nach `fork`, der Sohn davor.

( ) `fork()` liefert einen Wert zurück. Beim Sohn ist der Wert 0, beim Vater der Prozeßidentifier des gerade erzeugten Sohns.

( ) Kann man nicht feststellen (Eigenschaften eines Klon) und braucht man auch nicht, weil beide sich anschließend mit `exec` überlagern.

( ) An den Prozessidentifiern PID und PPID, die beide erhalten. Der mit der größeren PID ist der Sohn, weil später erzeugt.

Aufgabe 11

Was liefert die Kommandozeile `echo date | bash` ?

( ) eine Fehlermeldung der Art `bash: unexpected pipe`

( ) die Ausgabezeile `echo date`

( ) die Ausgabezeile `date`

( ) eine Datumsausgabe der Art `Mi 14. Jul 12:04:59 CEST 2010`

( ) keine Ausgabe, nur neues Prompt der `bash`

Aufgabe 12

Welches der folgenden Kommandos liefert immer den Namen des Heimatverzeichnisses (login directory)?

( ) `ls -d` `` `login` ``

( ) `ls -d`

( ) `ls -d .`

( ) `ls -d ~`

( ) `ls -d $HOME`

Aufgabe 13

In unserer Sammlung gefährlicher und dummer Dateinamen betrachten wir heute den Namen `"??"`. Kann man ihn erzeugen?

( ) Ja, kann man z. B. mit `touch \?\?` erzeugen.

( ) Ja, kann man z. B. mit `touch "??"` erzeugen.

( ) Ja, kann man in jedem Fall ganz locker mit `touch ??` erzeugen.

( ) Ja, kann man mit `touch '??'` erzeugen.

( ) Nein, `?` ist ein Shell-Metazeichen und damit in Dateinamen verboten.

Aufgabe 14

Ruft man das Kommando `which` mit einem Argument *cmd* auf, sucht es in den Verzeichnissen, die `$PATH` liefert, nach dem Programm *cmd*, das in einer Shell aufgerufen würde und gibt dessen Pfad aus, wenn cmd gefunden wird, sonst nichts. Genau einer der folgenden Aufrufe liefert keine Ausgabe. Welcher?

( ) `which which`

( ) `which man`

( ) `man man`

( ) `man which`

( ) `which cd`

Aufgabe 15

Wie im Kurs angedeutet, speichert man die verschlüsselten Passwörter nicht mehr in der für alle lesbaren Datei passwd, sondern in einer Datei shadow. Was gilt für diese Datei?

( ) Sie steht in `/etc`.

( ) Sie steht in `/bin`.

( ) Sie hat das s-Bit besetzt.

( ) Nur `root` kann sie lesen und schreiben.

( ) Die jetzt darin gespeicherten Passwörter sind nicht mehr verschlüsselt.

Aufgabe 16

Mit dem `wall`-Kommando können sie allen angemeldeten Benutzern eine Botschaft aufs Terminal zukommen lassen, die nicht `mesg n` gesetzt haben. Die Botschaft kann aus einer Datei kommen oder, wenn nichts angegeben ist, aus der Standardeingabe.

(a) Vervollständigen Sie das folgende Shell-Skript `mensa` mit einem *here*-Dokument.

mensa

```
(wall _________________

Sollen wir "$*" essen gehen? Danke!

Hunger

) && echo "$*" essen gehen.
```

(b) Rufen Sie `mensa` auf mit der Bitte, **in 15 Minuten** essen gehen zu wollen, oder **nach der Klausur**, oder **ins *** Lokal**, was Ihnen lieber ist, letzteres mit Vorsicht!

Prof. Dr. Lutz Wegner
Dipl.-Math. Dipl.-Inf. Sebastian Pape **26. Juli 2010**

Einführung in UNIX
Klausur zum Sommersemester 2010

MUSTERLÖSUNG

Name:.....Vorname:.....

Matr. Nr.:.....Studiengang:.....

Bearbeiten Sie alle Fragen! Bei Ankreuzaufgaben kann mehr als eine Antwort richtig sein. Hilfsmittel sind nicht zugelassen! Falls Sie für die Beantwortung der Aufgaben zusätzlichen Platz benötigen, verwenden Sie die Rückseiten.

Aufgabe	Punkte max.	Punkte erreicht
1	2	
2	2+2+2+2+2	
3	2	
4	4+2+2	
5	6	
6	2	
7	2	
8	2	
9	2	
10	2	
11	2	
12	2	
13	2	
14	2	
15	2	
16	2+2	
Summe	**52**	

Aufgabe 1

Für klassische Unix-Verhältnisse zeigt Linux ein ungewohnt geschwätziges, selbstzweiflerisches Verhalten bei der unten gezeigten Kommandozeile. Erläutern Sie den Sachverhalt.

```
fix@labserv:~$ mail fix </dev/null
No message, no subject; hope that's ok
fix@labserv:~$
```

/dev/null liefert sofort EOT und kein Betreff angegeben.

Aufgabe 2

Kommentieren Sie das Verhalten der Shell im Umgang mit Links auf Verzeichnisse unter Linux.

```
fix@labserv:~$ ln leerdir linkdir
ln: »leerdir«: harte Verknüpfung für Verzeichnisse nicht
erlaubt
fix@labserv:~$
```

(a) Hätte das laut Vorlesung und Skript funktionieren sollen?

Nein, keine harten Links auf Verzeichnisse.

```
fix@labserv:~$ ln -s leerdir linkdir
fix@labserv:~$ ls -l
insgesamt 84
...
drwxr-xr-x 2 fix fix  4096  8. Feb 15:32 leerdir
lrwxrwxrwx 1 fix fix     7  9. Jul 16:12 linkdir -> leerdir
...
fix@labserv:~$
```

(b) Was ist `linkdir` und warum ist die Dateigröße von `linkdir` 7 Bytes?

linkdir ist ein weicher Verweis, der auf leerdir zeigt. Als Datei enthält linkdir das Wort leerdir, das 7 Zeichen lang ist.

```
fix@labserv:~$ ls -l linkdir
lrwxrwxrwx 1 fix fix 7  9. Jul 16:12 linkdir -> leerdir
fix@labserv:~$
```

(c) Was zeigt `ls -l` hier? Was hätte man sich vielleicht gewünscht?

linkdir wird als Link in langer Form angezeigt. Weil linkdir auf ein Verzeichnis zeigt, hier leerdir, also ein alias für ein Verzeichnis ist, hätte man sich auch eine Auflistung des Inhalts von leerdir vorstellen können.

```
fix@labserv:~$ echo hallo >linkdir/hallofile
fix@labserv:~$ ls -l leerdir
insgesamt 4
-rw-r--r-- 1 fix fix 6  9. Jul 16:13 hallofile
fix@labserv:~$ ls -l linkdir/.
insgesamt 4
-rw-r--r-- 1 fix fix 6  9. Jul 16:13 hallofile
fix@labserv:~$
```

(d) Funktioniert `linkdir` hier wie ein Verzeichnis? Kurze Begründung was die Shell macht!

Ja, die Shell ersetzt an den Stellen, so der Softlink im Pfad auftaucht, den Link durch den referenzierten Namen und prüft dann, ob das einen gültigen Pfad ergibt.

```
fix@labserv:~$ cd linkdir
fix@labserv:~/linkdir$ ls
hallofile
fix@labserv:~/linkdir$ pwd
/home/fix/linkdir
fix@labserv:~/linkdir$
```

(e) Kann man `linkdir` zum Arbeitsverzeichnis machen?

Ja, Begründung wie oben.

Aufgabe 3

Bei einigen Teilnehmern ist in `/etc/passwd` an der Stelle für die login-Shell `/bin/false` eingetragen, also keine Shell, z. B. beim Eintrag
`postfix:x:111:119::/var/spool/postfix:/bin/false`
Welche Aussage ist richtig, bzw. welche sind richtig?

(X) Liefert beim versuchten Aufruf sofort einen Fehlerstatuscode ab.

( ) Die eigentliche Shell wird beim Systemstart erst eingetragen.

( ) `/bin/false` ist tatsächlich eine Shell, die sog. Alias-Shell (free alias shell extension)

( ) Der Eintrag ist eigentlich irrelevant, er könnte auch `/usr/bin/vi` sein.

( ) Selbst der Teilnehmer `root` hat dort diesen Eintrag `/bin/false`.

Aufgabe 4

In der Vorlesung und im Skript werden die sog. *compound commands* der Shell mit Ausnahme der Untershell nicht behandelt. So wertet `((`*expression*`))` einen arithmetischen Ausdruck, also *expression* aus. Ist das Ergebnis der Auswertung ungleich Null liefert es einen Statuswert 0 zurück, sonst liefert es 1. Mit dem `let`-Kommando kann man den Wert des Ausdrucks einer Variablen zuweisen. Dies verwenden wir im folgenden Skript `mycalc`.

mycalc

```
if [ $# -lt 1 ]; then
  echo "Keine Argumente - keine Auswertung. So isses."
elif (( $* )); then
  let x="$*"
  echo "$* = $x"
else
  echo "$* = 0 oder kein arithmetischer Ausdruck"
fi
```

(a) Füllen Sie die Lücken in `mycalc` unter Berücksichtigung der folgenden Aufrufe.

```
fix@labserv:~/aushang$ ../mycalc
Keine Argumente - keine Auswertung. So isses.
fix@labserv:~/aushang$ ../mycalc 3 + 4
3 + 4 = 7
fix@labserv:~/aushang$ ../mycalc 4 - 4
4 - 4 = 0 oder kein arithmetischer Ausdruck
fix@labserv:~/aushang$ ../mycalc 4 4
../mycalc: line 3: ((: 4 4 : syntax error in expression (error
token is "4 ")
4 4 = 0 oder kein arithmetischer Ausdruck
```

(b) Erläutern Sie die folgende Ausgabe!

```
fix@labserv:~/aushang$ ../mycalc 4 * 4
../mycalc: line 3: ((: 4 nullo tcltk 4 : syntax error in
expression (error token is "nullo tcltk 4 ")
4 nullo tcltk 4 = 0 oder kein arithmetischer Ausdruck
```

Wildcard-Symbol * ist nicht maskiert und wird durch die Einträge im Arbeitsverzeichnis ersetzt.

(c) Wie muss der Aufruf für eine Multiplikation korrekt lauten?

```
fix@labserv:~/aushang$ ../mycalc 4 \* 4
4 * 4 = 16
```

auch "*" oder '*'

Aufgabe 5

Unser Shell-Skript `cleanup`, das zum Entfernen leerer Normaldateien auffordert, liefert bei Aufruf im Arbeitsverzeichnis `aushang` die folgende Ausgabe. Ergänzen Sie die Lücken im Shell-Skript. Dateien, die mit Punkt beginnen, brauchen nicht berücksichtigt zu werden.

```
fix@labserv:~/aushang$ ls -l
insgesamt 8
drwxr-xr-x 2 fix fix 4096 14. Jul 11:27 MeinKatalog
-rw-r--r-- 1 fix fix    0 12. Jul 14:24 nullo
-rw-r--r-- 1 fix fix  100 20. Nov 2009  tcltk
fix@labserv:~/aushang$ ../cleanup
MeinKatalog keine leere Datei
rm: reguläre leere Datei »nullo« entfernen? n
tcltk keine leere Datei
fix@labserv:~/aushang$
```

cleanup

```
for name in ‘ls‘
do
  if [ -f "$name" -a ! -s "$name" ]
  then
    rm -i $name
  else
    echo $name keine leere Datei
  fi
done
```

Aufgabe 6

Im Skript und auch im Kurs wird erwähnt, dass das Ändern des Besitzers (und der Besitzergruppe) einer Datei mit `chown` in den meisten neueren Unix- und Linux-Versionen nur noch dem Super-User vorbehalten ist. Aber wie kann ein normaler Anwender für seine Arbeit weiterhin anderen Gruppenrechte einräumen? Funktioniert das `chgrp`-Kommando weiterhin?

(X) Ja, aber nur der Besitzer einer Datei kann die Gruppe dieser Datei auf jede andere Gruppe ändern, in der er Mitglied ist.

( ) Ja, aber nur der Besitzer einer Datei kann die Gruppe dieser Datei auf jede andere Gruppe ändern, auch wenn er nicht Mitglied ist.

( ) Ja, grundsätzlich kann sogar jeder Anwender die Besitzergruppe einer Datei ändern, auch wenn er nicht deren Besitzer ist.

( ) Nein, Besitzergruppen für Dateien, Verzeichnisse und Geräte gibt es nicht mehr. Es wird nur noch aus Kompatibilitätsgründen eine Gruppe `group` eingetragen, der aber alle angehören.

( ) Nein, das Kommando zum Ändern der Besitzergruppe heißt `newgrp`.

Aufgabe 7

Welche Aussagen zum i-node (bzw. der i-Nummer) im UNIX-Dateisystem sind richtig?

(X) Jede Normaldatei und jedes Verzeichnis hat genau einen i-node.

(X) In den Verzeichnissen sind beim Dateinamen die zugehörige i-Nummer eingetragen (wegen der harten Links ggf. bei mehreren Namen die selbe i-Nummer).

(X) Im i-node sind unter anderem der owner-id (Besitzer-Id) und die für die Datei gesetzten Rechte eingetragen.

( ) Gehört der i-node zu einem Verzeichnis, ist im i-node selbst die i-Nummer des Vaterverzeichnisses und die i-Nummer des eigenen i-node eingetragen (wegen `.` und `..`).

( ) Wächst eine Datei über eine gewisse Größe (derzeit 2 MB bei Linux), muss auch der i-node wachsen, weil er die Adressen **aller** Datenblöcke aufnimmt.

Aufgabe 8

Kann man sich in einem laufenden Shell-Skript die Prozessnummer (process id) des ausführenden Prozesses anzeigen lassen?

(X) Ja, z. B. mit `echo $$`. Man erhält den Prozessidentifier des Shell,

( ) Ja, z. B. mit echo `$PPID`. Auch so erhält man den Prozessidentifier der ausführenden Shell.

( ) Nein, ein Shell-Skript ist kein executable und hat zur Laufzeit keinen Prozessidentifier.

( ) Nein, dazu müßte man zwingend das Skript in einer Unter-Untershell laufen lassen.

( ) Nur wenn das t-Bit gesetzt ist.

Aufgabe 9

Wenn zwei oder mehr Kommandos über eine Pipe verbunden werden, z. B. bei `sort` *xyz* | `uniq | lpd`, dann ...

( ) ... können die hinteren Kommandos erst starten, wenn die vorderen fertig sind.

( ) ... findet nach jedem weitergegebenen Byte (Zeichen) ein Prozessorwechsel zum nächsten Prozess statt (nach dem letzten wieder zurück zum ersten Prozess in der Pipe).

( ) ... darf keines der beteiligten Kommandos ein Shell-Skript sein.

( ) ... ist bei allen beteiligten Kommandos die Standardfehlerausgabe in die Standardausgabe umgeleitet.

(X) ... schreibt die Standardausgabe des Kommandos links vom Pipesymbol in die Standardeingabe des Kommandos rechts davon.

Aufgabe 10

Nach einem `fork()`-Systemaufruf entstehen zwei identische Prozessabbilder. Woran erkennen die beiden Prozesse, wer Vater und wer Sohn ist?

( ) Am Alter. Die Prozesse fragen die registrierte Laufzeit ab (`ERTIME`, elapsed running time), die beim Sohn 0 und beim Vater > 0 ist (wegen des `fork`-Aufrufs).

( ) Am Befehlszeiger. Der Vater steht nach `fork`, der Sohn davor.

(X) `fork()` liefert einen Wert zurück. Beim Sohn ist der Wert 0, beim Vater der Prozeßidentifier des gerade erzeugten Sohns.

( ) Kann man nicht feststellen (Eigenschaften eines Klon) und braucht man auch nicht, weil beide sich anschließend mit `exec` überlagern.

( ) An den Prozessidentifiern PID und PPID, die beide erhalten. Der mit der größeren PID ist der Sohn, weil später erzeugt.

Aufgabe 11

Was liefert die Kommandozeile `echo date | bash` ?

( ) eine Fehlermeldung der Art `bash: unexpected pipe`

( ) die Ausgabezeile `echo date`

( ) die Ausgabezeile `date`

(X) eine Datumsausgabe der Art `Mi 14. Jul 12:04:59 CEST 2010`

( ) keine Ausgabe, nur neues Prompt der `bash`

Aufgabe 12

Welches der folgenden Kommandos liefert immer den Namen des Heimatverzeichnisses (login directory)?

( ) `ls -d` `` `login` ``

( ) `ls -d`

( ) `ls -d .`

(X) `ls -d ~`

(X) `ls -d $HOME`

Aufgabe 13

In unserer Sammlung gefährlicher und dummer Dateinamen betrachten wir heute den Namen `"??"`. Kann man ihn erzeugen?

(X) Ja, kann man z. B. mit `touch \?\?` erzeugen.

(X) Ja, kann man z. B. mit `touch "??"` erzeugen.

( ) Ja, kann man in jedem Fall ganz locker mit `touch ??` erzeugen.

(X) Ja, kann man mit `touch '??'` erzeugen.

( ) Nein, `?` ist ein Shell-Metazeichen und damit in Dateinamen verboten.

Aufgabe 14

Ruft man das Kommando `which` mit einem Argument *cmd* auf, sucht es in den Verzeichnissen, die `$PATH` liefert, nach dem Programm *cmd*, das in einer Shell aufgerufen würde und gibt dessen Pfad aus, wenn cmd gefunden wird, sonst nichts. Genau einer der folgenden Aufrufe liefert keine Ausgabe. Welcher?

( ) `which which`

( ) `which man`

( ) `man man`

( ) `man which`

(X) `which cd`

Aufgabe 15

Wie im Kurs angedeutet, speichert man die verschlüsselten Passwörter nicht mehr in der für alle lesbaren Datei passwd, sondern in einer Datei shadow. Was gilt für diese Datei?

(X) Sie steht in `/etc`.

( ) Sie steht in `/bin`.

( ) Sie hat das s-Bit besetzt.

(X) Nur `root` kann sie lesen und schreiben.

( ) Die jetzt darin gespeicherten Passwörter sind nicht mehr verschlüsselt.

Aufgabe 16

Mit dem `wall`-Kommando können sie allen angemeldeten Benutzern eine Botschaft aufs Terminal zukommen lassen, die nicht `mesg n` gesetzt haben. Die Botschaft kann aus einer Datei kommen oder, wenn nichts angegeben ist, aus der Standardeingabe.

(a) Vervollständigen Sie das folgende Shell-Skript `mensa` mit einem *here*-Dokument.

mensa

```
(wall <<Hunger
Sollen wir "$*" essen gehen? Danke!
Hunger
) && echo "$*" essen gehen.
```

(b) Rufen Sie `mensa` auf mit der Bitte, **in 15 Minuten** essen gehen zu wollen, oder **nach der Klausur**, oder **ins *** Lokal**, was Ihnen lieber ist, letzteres mit Vorsicht!

```
mensa "in 15 Minuten"
```

Prof. Dr. Lutz Wegner
Dipl.-Math. Dipl.-Inf. Sebastian Pape **18. Februar 2011**

Einführung in UNIX
Klausur zum Wintersemester 2010/11

Name:.....Vorname:.....

Matr. Nr.:.....Studiengang:.....

Bearbeiten Sie alle Fragen! Bei Ankreuzaufgaben kann mehr als eine Antwort richtig sein. Hilfsmittel sind nicht zugelassen! Falls Sie für die Beantwortung der Aufgaben zusätzlichen Platz benötigen, verwenden Sie die Rückseiten.

Aufgabe	Punkte max.	Punkte erreicht
1	2	
2	2	
3	2	
4	2	
5	2	
6	4	
7	2	
8	2	
9	2	
10	2	
11	2	
12	2	
13	2	
14	2	
15	2	
16	2	
17	2	
Summe	**36**	

Aufgabe 1

Wenn man zwangsweise einen Prozess beenden muss, was unterscheidet den Aufruf mit dem Signal `SIGTERM` (Default) von dem Aufruf mit `SIGKILL` ?

( ) SIGTERM kann vom Prozess abgefangen werden und erlaubt ihm eine geordneten Abbruch mit Schließen von Dateien.

( ) SIGTERM läßt Unterprozesse am Leben, die man vielleicht noch braucht.

( ) kill mit dem Signal SIGKILL kann nur der Superuser aufrufen.

( ) SIGKILL funktioniert zwar immer, erzeugt aber zwangsweise einen Zombie-Prozess, der erst beim Runterfahren der Anlage verschwindet.

( ) SIGKILL beendet immer alle Prozesse des Aufrufers, man muss also keinen Prozess-identifier angeben.

Aufgabe 2

Was `/dev/null` anbetrifft, warum kann ich die Datei `ankuendigung` nicht mit `mv` im Biteimer verschwinden lassen, obwohl für `/dev/null` das Schreibrecht für alle gesetzt ist?

```
fix@labserv:~$ mv ankuendigung /dev/null
mv: Verschieben zwischen Geräten fehlgeschlagen:
    »ankuendigung« zu »/dev/null«; kann Ziel nicht
    entfernen: Keine Berechtigung
fix@labserv:~$
```

Aufgabe 3

Ergänzen Sie die Lücken für die bedingte Verbindung zweier Kommandos!

```
fix@labserv:~$ ecco _____ date
-bash: ecco: command not found
fix@labserv:~$ ecco _____ date
-bash: ecco: command not found
Do 20. Jan 14:11:56 CET 2011
fix@labserv:~$
```

Aufgabe 4

Warum erzeugt die Kommandozeile `ls -l | pr -3` diese unschöne Ausgabe?

```
2011-01-25 16:30                                                Seite 1


insgesamt 64            -r--rw-rw- 1 fix users  -rwxr-xr-x 1 fix fix
-rw-r--r-- 1 fix fix    -rwxr-xr-x 1 fix fix    -rwxr-xr-x 1 fix fix
-rw-r--r-- 1 fix fix    -rwxr-xr-x 1 fix fix    lrwxrwxrwx 1 fix fix
-rw-r--r-- 1 fix users  -rw------- 1 fix fix    lrwxrwxrwx 1 fix fix
-rwxr-xr-x 1 fix fix    -rwxr-xr-x 1 fix fix    drwxr-xr-x 2 fix fix
-rw-r--r-- 1 fix users  drwxr-xr-x 2 fix fix    -rwxr-xr-x 1 fix fix
drwxr-xr-x 2 fix fix
```

Kurze Antwort:

Aufgabe 5

Könnte es unter Linux ein `shred`-Kommando geben, das eine zu löschende Datei mehrmals mit Zufallswerten überschreibt, um den Inhalt auch für anspruchsvolle Plattenwiederherstellsysteme unlesbar zu machen?

( ) Ja, könnte es geben, würde aber nur funktionieren, wenn das verwendete Unix-Dateisystem grundsätzlich Daten am Ort überschreibt und kein Journal, Log oder Schnappschüsse zur Wiederherstellung bereithält.

( ) Ist unnötig, `rm` alleine leistet das auch.

( ) Nein, kann es wegen der i-nodes nicht geben.

( ) Nein, weil man leicht ein Programm `unshred` schreiben könnte, das das Überschreiben rückgängig macht.

Aufgabe 6

Wie könnte ein möglichst einfaches here-Dokument (Shell-Skript) aussehen, das die folgende Ausgaben produziert? Antwort bitte auf der Rückseite.

```
fix@labserv:~$ ./here pi pa po
Dieser Text stammt aus
dem Shell-Skript selbst.
Die Argumente des Aufrufs waren pi pa po
Alles klar?
fix@labserv:~$ ./here Hallo Leute
Dieser Text stammt aus
dem Shell-Skript selbst.
Die Argumente des Aufrufs waren Hallo Leute
Alles klar?
fix@labserv:~$
```

Aufgabe 7

Mit `ln -s` kann man einen Softlink sogar auf Verzeichnisse einrichten. Das wollen wir mit

```
fix@labserv:/home/fix$ ln -s ../.. '...'
```

nutzen, um ein Verzeichnis „`...`“ einzurichten. Unseren Freunden reden wir ein, mit `ls ...` würde immer der Inhalt des Vor-Vorgängerverzeichnis angezeigt.

( ) Ja, funktioniert aber nur in dem Verzeichnis, wo „`...`“ eingetragen ist.

( ) Ja, funktioniert überall, weil es so der Shell bekanntgemacht wurde.

( ) Nein, „`...`“ als Verzeichnisname geht schon, aber das mit `ln` und `ls` funktioniert nicht, auch nicht für das gegenwärtige Verzeichnis, in dem „`...`“ eingetragen ist.

( ) Nein, „`...`“ ist weder als Verzeichnisname zugelassen noch funktioniert das mit `ln` und `ls` in der gezeigten Form.

Aufgabe 8

Der `init`-Prozess ...

( ) ... erbt alle Zombie-Prozesse.

( ) ... erbt alle Waisenkinder.

( ) ... kontrolliert alle mit dem `at`-Kommando gestarteten Prozesse.

( ) ... wird automatisch Besitzer aller Hintergrundprozesse.

( ) ... ist der einzige Prozess, der kein `fork()` kann.

Aufgabe 9

Das Textsegment eines Prozesses im virtuellen Adressraum wird zum Überschreiben freigegeben (gelöscht), wenn

( ) der Platz knapp wird, unabhängig davon, wie viele Prozesse es gerade nutzen.

( ) kein Prozess es mehr nutzt.

( ) der Link-Zähler im i-Knoten der Programmdatei, von der das Textsegment stammt, auf 0 heruntergezählt wurde.

( ) ein Prozess in das Textsegment schreibt, das Schreibrecht dafür für alle (others) gesetzt war oder dieser überschreibende Prozess root gehört.

Aufgabe 10

Das id-Kommando zeigt neben der User-Id auch die effektive User-Id (EUID), wenn diese von der UID abweicht. Mit `cp /usr/bin/id .` hat sich `fix` dieses Kommando, das ein Binary ist, in sein Arbeitsverzeichnis kopiert. Nach einigen weiteren Schritten kann er das folgende Ergebnis produzieren, das die Unterscheidung zeigt:

```
lwegner@labserv:/home/fix$ ./id

uid=59802(lwegner) gid=5000(uniks) euid=1006(fix)
Gruppen=100(users),117(fuse),1005(students12),1006(student
s1),5000(uniks)

lwegner@labserv:/home/fix$
```

Was ist richtig?

( ) An der eigenen Kopie von `id` wurde das SUID-Bits gesetzt.

( ) Es wurde ein login als `lwegner` gemacht (z. B. mit `su`).

( ) Das Heimatverzeichnis von `fix`, wo jetzt die Kopie von `id` steht, war betretbar oder wurde betretbar für `lwegner` gemacht.

( ) Statt einer lokalen Kopie des `id`-Kommandos hätte man auch ein Shell-Skript, sagen wir `myid`, schreiben können, das `/usr/bin/id` aufruft. Am Shell-Skript `myid` müsste das SUID-Bit gesetzt werden.

( ) Eigentlich zeigt nach einem Identitätswechsel mit `su` das Kommando `id` sowieso immer unterschiedliche UID und EUID an.

Aufgabe 11

Betrachten Sie das folgende Shell-Skript `anschrift`.

```
echo Bitte machen Sie die folgenden Angaben.
echo -n Titel:" "; read titel
echo -n Nachname Vorname:" "; read vname nname
echo Sehr geehrte/r Frau/Herr $titel $nname
```

Der Aufruf und die Anwendereingaben (kursiv) lauten wie folgt. Ergänzen Sie die Ausgabe!

```
fix@labserv:~$ ./anschrift

Bitte machen Sie die folgenden Angaben.

Titel: Prof. Dr.

Nachname Vorname: Immanuel E. Unrat
```

_____

Aufgabe 12

Welches der folgenden Kommandos setzt das Arbeitsverzeichnisses auf das Heimatverzeichnis?

( ) `cd `pwd``

( ) `cd`

( ) `cd .`

( ) `cd ..`

( ) `cd $HOME`

Aufgabe 13

In unserer Sammlung gefährlicher und dummer Dateinamen betrachten wir heute den Namen, der nur aus dem Komma „`,`“ besteht. Kann man ihn erzeugen?

( ) Ja, kann man z. B. mit `touch \,` erzeugen.

( ) Ja, kann man z. B. mit `touch ","` erzeugen.

( ) Ja, kann man sogar ganz locker mit `touch ,` erzeugen.

( ) Ja, kann man mit `touch ','` erzeugen.

( ) Nein, das Komma ist als Dateiname verboten.

Aufgabe 14

Teilnehmer `fix` will mit `chgrp users ankuendigung` die Gruppe `users` zum Gruppenbesitzer der Datei `ankuendigung` machen. Was ist richtig?

( ) Die Datei muss `fix` gehören.

( ) `fix` muss Mitglied der Gruppe `users` sein.

( ) `fix` muss vorher mit `newgrp` die aktuelle Gruppenkennung ändern.

( ) Geht nicht mit `chgrp` sodern mit `chown`.

( ) Nachträgliches ändern des Gruppenbesitzers geht gar nicht mehr.

Aufgabe 15

Ergänzen Sie die Lücke!

```
fix@labserv:~$ which _____________
/usr/bin/which
fix@labserv:~$
```

Aufgabe 16

Welche Aussage ist, bzw. welche Aussagen sind in Zusammenhang mit dem `read`-Kommando richtig?

( ) `read` ist ein Spezialkommando (in der Shell ausgeführtes Kommando), weil sonst die Rückgabe der Variablenwerte nicht funktionieren würde.

( ) `read` kann nur vom Terminal lesen, nicht z. B. aus einer Pipe.

( ) Die Belegung von `IFS` bestimmt, wie die Eingabezeile in Wörter getrennt wird.

( ) Paßt bei der Eingabe für ein `read` am Terminal die Eingabe nicht in eine Zeile, kann man mit dem backslash-Zeichen „\“ die Bedeutung des newline-Zeichens am Ende der ersten Zeile aufheben und in der zweiten Zeile weiterschreiben.

Aufgabe 17

Ihr Unix-System wird vermutlich Kommandos enthalten, von denen Sie noch nie gehört haben. Worauf können Sie sich bei Einhaltung der Unix-Konventionen trotzdem verlassen?

( ) Bei erfolgreicher Beendigung des Kommandos wird Exitstatus 0 zurückgeliefert.

( ) Fehlerausgaben kommen auf der Standardfehlerausgabe (file descriptor 2) raus.

( ) Wenn das Kommando eine größere Eingabe erwartet, dann liest das Kommando unterschiedslos vom Terminal, einer Datei oder aus einer Pipe.

( ) Das Kommando wird nie eine Datei löschen, ohne vorher interaktiv die Erlaubnis dafür einzuholen.

( ) Alle Optionen sind bei allen Kommandos immer gleich und mit gleicher Bedeutung, z. B. bedeutet `-r` immer rekursive Ausführung.

ENDE DER KLAUSUR

Prof. Dr. Lutz Wegner
Dipl.-Math. Dipl.-Inf. Sebastian Pape **18. Februar 2011**

Einführung in UNIX
Klausur zum Wintersemester 2010/11

Name:.....Musterlösung.....**Vorname:**.....

Matr. Nr.:.....**Studiengang:**.....

Bearbeiten Sie alle Fragen! Bei Ankreuzaufgaben kann mehr als eine Antwort richtig sein. Hilfsmittel sind nicht zugelassen! Falls Sie für die Beantwortung der Aufgaben zusätzlichen Platz benötigen, verwenden Sie die Rückseiten.

Aufgabe	Punkte max.	Punkte erreicht
1	2	
2	2	
3	2	
4	2	
5	2	
6	4	
7	2	
8	2	
9	2	
10	2	
11	2	
12	2	
13	2	
14	2	
15	2	
16	2	
17	2	
Summe	**36**	

Aufgabe 1

Wenn man zwangsweise einen Prozess beenden muss, was unterscheidet den Aufruf mit dem Signal `SIGTERM` (Default) von dem Aufruf mit `SIGKILL` ?

(X) SIGTERM kann vom Prozess abgefangen werden und erlaubt ihm eine geordneten Abbruch mit Schließen von Dateien.

( ) SIGTERM läßt Unterprozesse am Leben, die man vielleicht noch braucht.

( ) kill mit dem Signal SIGKILL kann nur der Superuser aufrufen.

( ) SIGKILL funktioniert zwar immer, erzeugt aber zwangsweise einen Zombie-Prozess, der erst beim Runterfahren der Anlage verschwindet.

( ) SIGKILL beendet immer alle Prozesse des Aufrufers, man muss also keinen Prozess-identifier angeben.

Aufgabe 2

Was `/dev/null` anbetrifft, warum kann ich die Datei `ankuendigung` nicht mit `mv` im Biteimer verschwinden lassen, obwohl für `/dev/null` das Schreibrecht für alle gesetzt ist?

```
fix@labserv:~$ mv ankuendigung /dev/null
mv: Verschieben zwischen Geräten fehlgeschlagen:
    »ankuendigung« zu »/dev/null«; kann Ziel nicht
    entfernen: Keine Berechtigung
fix@labserv:~$
```

Bei mv würde die Spezialdatei *`/dev/null`* durch die Datei *`ankuendigung`* ersetzt, das erlauben aber die Dateirechte in *`/dev`* nicht.

Aufgabe 3

Ergänzen Sie die Lücken für die bedingte Verbindung zweier Kommandos!

```
fix@labserv:~$ ecco _&&_ date
-bash: ecco: command not found
fix@labserv:~$ ecco __||__ date
-bash: ecco: command not found
Do 20. Jan 14:11:56 CET 2011
fix@labserv:~$
```

Aufgabe 4

Warum erzeugt die Kommandozeile `ls -l | pr -3` diese unschöne Ausgabe?

```
2011-01-25 16:30                                                Seite 1


insgesamt 64            -r--rw-rw- 1 fix users  -rwxr-xr-x 1 fix fix
-rw-r--r-- 1 fix fix    -rwxr-xr-x 1 fix fix    -rwxr-xr-x 1 fix fix
-rw-r--r-- 1 fix fix    -rwxr-xr-x 1 fix fix    lrwxrwxrwx 1 fix fix
-rw-r--r-- 1 fix users  -rw------- 1 fix fix    lrwxrwxrwx 1 fix fix
-rwxr-xr-x 1 fix fix    -rwxr-xr-x 1 fix fix    drwxr-xr-x 2 fix fix
-rw-r--r-- 1 fix users  drwxr-xr-x 2 fix fix    -rwxr-xr-x 1 fix fix
drwxr-xr-x 2 fix fix
```

Kurze Antwort:

Bei der Ausgabe in drei Spalten kürzt *pr* die Zeilen, die länger als die Spaltenbreite sind (kein Umbruch)

Aufgabe 5

Könnte es unter Linux ein `shred`-Kommando geben, das eine zu löschende Datei mehrmals mit Zufallswerten überschreibt, um den Inhalt auch für anspruchsvolle Plattenwiederherstellsysteme unlesbar zu machen?

(X) Ja, könnte es geben, würde aber nur funktionieren, wenn das verwendete Unix-Dateisystem grundsätzlich Daten am Ort überschreibt und kein Journal, Log oder Schnappschüsse zur Wiederherstellung bereithält.

( ) Ist unnötig, `rm` alleine leistet das auch.

( ) Nein, kann es wegen der i-nodes nicht geben.

( ) Nein, weil man leicht ein Programm `unshred` schreiben könnte, das das Überschreiben rückgängig macht.

Aufgabe 6

Wie könnte ein möglichst einfaches here-Dokument (Shell-Skript) aussehen, das die folgende Ausgaben produziert? Antwort bitte auf der Rückseite.

```
fix@labserv:~$ ./here pi pa po
Dieser Text stammt aus
dem Shell-Skript selbst.
Die Argumente des Aufrufs waren pi pa po
Alles klar?
fix@labserv:~$ ./here Hallo Leute
Dieser Text stammt aus
dem Shell-Skript selbst.
Die Argumente des Aufrufs waren Hallo Leute
Alles klar?
fix@labserv:~$
```

```
cat <<EOF
Dieser Text stammt aus
dem Shell-Skript selbst.
Die ... waren $*
Alles klar?
EOF
```

Aufgabe 7

Mit `ln -s` kann man einen Softlink sogar auf Verzeichnisse einrichten. Das wollen wir mit

```
fix@labserv:/home/fix$ ln -s ../.. '...'
```

nutzen, um ein Verzeichnis „`...`“ einzurichten. Unseren Freunden reden wir ein, mit `ls ...` würde immer der Inhalt des Vor-Vorgängerverzeichnis angezeigt.

(X) Ja, funktioniert aber nur in dem Verzeichnis, wo „`...`“ eingetragen ist.

( ) Ja, funktioniert überall, weil es so der Shell bekanntgemacht wurde.

( ) Nein, „`...`“ als Verzeichnisname geht schon, aber das mit `ln` und `ls` funktioniert nicht, auch nicht für das gegenwärtige Verzeichnis, in dem „`...`“ eingetragen ist.

( ) Nein, „`...`“ ist weder als Verzeichnisname zugelassen noch funktioniert das mit `ln` und `ls` in der gezeigten Form.

Aufgabe 8

Der `init`-Prozess ...

( ) ... erbt alle Zombie-Prozesse.

(X) ... erbt alle Waisenkinder.

( ) ... kontrolliert alle mit dem `at`-Kommando gestarteten Prozesse.

( ) ... wird automatisch Besitzer aller Hintergrundprozesse.

( ) ... ist der einzige Prozess, der kein `fork()` kann.

Aufgabe 9

Das Textsegment eines Prozesses im virtuellen Adressraum wird zum Überschreiben freigegeben (gelöscht), wenn

( ) der Platz knapp wird, unabhängig davon, wie viele Prozesse es gerade nutzen.

(X) kein Prozess es mehr nutzt.

( ) der Link-Zähler im i-Knoten der Programmdatei, von der das Textsegment stammt, auf 0 heruntergezählt wurde.

( ) ein Prozess in das Textsegment schreibt, das Schreibrecht dafür für alle (others) gesetzt war oder dieser überschreibende Prozess root gehört.

Aufgabe 10

Das id-Kommando zeigt neben der User-Id auch die effektive User-Id (EUID), wenn diese von der UID abweicht. Mit `cp /usr/bin/id .` hat sich `fix` dieses Kommando, das ein Binary ist, in sein Arbeitsverzeichnis kopiert. Nach einigen weiteren Schritten kann er das folgende Ergebnis produzieren, das die Unterscheidung zeigt:

```
lwegner@labserv:/home/fix$ ./id

uid=59802(lwegner) gid=5000(uniks) euid=1006(fix)
Gruppen=100(users),117(fuse),1005(students12),1006(student
s1),5000(uniks)

lwegner@labserv:/home/fix$
```

Was ist richtig?

(X) An der eigenen Kopie von `id` wurde das SUID-Bits gesetzt.

(X) Es wurde ein login als `lwegner` gemacht (z. B. mit `su`).

(X) Das Heimatverzeichnis von `fix`, wo jetzt die Kopie von `id` steht, war betretbar oder wurde betretbar für `lwegner` gemacht.

( ) Statt einer lokalen Kopie des `id`-Kommandos hätte man auch ein Shell-Skript, sagen wir `myid`, schreiben können, das `/usr/bin/id` aufruft. Am Shell-Skript `myid` müsste das SUID-Bit gesetzt werden.

( ) Eigentlich zeigt nach einem Identitätswechsel mit `su` das Kommando `id` sowieso immer unterschiedliche UID und EUID an.

Aufgabe 11

Betrachten Sie das folgende Shell-Skript `anschrift`.

```
echo Bitte machen Sie die folgenden Angaben.
echo -n Titel:" "; read titel
echo -n Nachname Vorname:" "; read vname nname
echo Sehr geehrte/r Frau/Herr $titel $nname
```

Der Aufruf und die Anwendereingaben (kursiv) lauten wie folgt. Ergänzen Sie die Ausgabe!

```
fix@labserv:~$ ./anschrift

Bitte machen Sie die folgenden Angaben.

Titel: Prof. Dr.

Nachname Vorname: Immanuel E. Unrat

Sehr geehrte/r Frau/Herr Prof. Dr. E. Unrat
```

Aufgabe 12

Welches der folgenden Kommandos setzt das Arbeitsverzeichnisses auf das Heimatverzeichnis?

( ) `cd `pwd``

(X) `cd`

( ) `cd .`

( ) `cd ..`

(X) cd $HOME

Aufgabe 13

In unserer Sammlung gefährlicher und dummer Dateinamen betrachten wir heute den Namen, der nur aus dem Komma „`,`“ besteht. Kann man ihn erzeugen?

(X) Ja, kann man z. B. mit `touch \,` erzeugen.

(X) Ja, kann man z. B. mit `touch ","` erzeugen.

(X) Ja, kann man sogar ganz locker mit `touch ,` erzeugen.

(X) Ja, kann man mit `touch ','` erzeugen.

( ) Nein, das Komma ist als Dateiname verboten.

Aufgabe 14

Teilnehmer `fix` will mit `chgrp users ankuendigung` die Gruppe `users` zum Gruppenbesitzer der Datei `ankuendigung` machen. Was ist richtig?

(X) Die Datei muss `fix` gehören.

(X) `fix` muss Mitglied der Gruppe `users` sein.

( ) `fix` muss vorher mit `newgrp` die aktuelle Gruppenkennung ändern.

( ) Geht nicht mit `chgrp` sodern mit `chown`.

( ) Nachträgliches ändern des Gruppenbesitzers geht gar nicht mehr.

Aufgabe 15

Ergänzen Sie die Lücke!

```
fix@labserv:~$ which which
/usr/bin/which
fix@labserv:~$
```

Aufgabe 16

Welche Aussage ist, bzw. welche Aussagen sind in Zusammenhang mit dem `read`-Kommando richtig?

(X) `read` ist ein Spezialkommando (in der Shell ausgeführtes Kommando), weil sonst die Rückgabe der Variablenwerte nicht funktionieren würde.

( ) `read` kann nur vom Terminal lesen, nicht z. B. aus einer Pipe.

(X) Die Belegung von `IFS` bestimmt, wie die Eingabezeile in Wörter getrennt wird.

(X) Paßt bei der Eingabe für ein `read` am Terminal die Eingabe nicht in eine Zeile, kann man mit dem backslash-Zeichen „\“ die Bedeutung des newline-Zeichens am Ende der ersten Zeile aufheben und in der zweiten Zeile weiterschreiben.

Aufgabe 17

Ihr Unix-System wird vermutlich Kommandos enthalten, von denen Sie noch nie gehört haben. Worauf können Sie sich bei Einhaltung der Unix-Konventionen trotzdem verlassen?

(X) Bei erfolgreicher Beendigung des Kommandos wird Exitstatus 0 zurückgeliefert.

(X) Fehlerausgaben kommen auf der Standardfehlerausgabe (file descriptor 2) raus.

(X) Wenn das Kommando eine größere Eingabe erwartet, dann liest das Kommando unterschiedslos vom Terminal, einer Datei oder aus einer Pipe.

( ) Das Kommando wird nie eine Datei löschen, ohne vorher interaktiv die Erlaubnis dafür einzuholen.

( ) Alle Optionen sind bei allen Kommandos immer gleich und mit gleicher Bedeutung, z. B. bedeutet `-r` immer rekursive Ausführung.

ENDE DER KLAUSUR

Prof. Dr. Lutz Wegner **18. Juli 2011**

Einführung in UNIX
Klausur zum Sommersemester 2011

Name:.....Vorname:.....

Matr. Nr.:.....Studiengang:.....

Bearbeiten Sie alle Fragen! Bei Ankreuzaufgaben kann mehr als eine Antwort richtig sein. Hilfsmittel sind nicht zugelassen! Falls Sie für die Beantwortung der Aufgaben zusätzlichen Platz benötigen, verwenden Sie die Rückseiten. Bearbeitungszeit 90 Minuten.

Aufgabe	Punkte max.	Punkte erreicht
1	2	
2	2	
3	2	
4	2	
5	2	
6	2	
7	2	
8	2	
9	2+2+2+2+2+2+2	
10	2	
11	2	
12	2	
13	2	
14	2	
Summe	**40**	

Aufgabe 1

Prof. Fix will sich selbst eine Testmail schicken mit dem Betreff „Testmail“, ist sich aber nicht ganz sicher, wo der Betreff hingehört. Was ergibt sich aus dem folgenden Ablauf?

```
fix@labserv:~$ mail fix Testmail
Subject: test
Hallo Leute!
Cc:
fix@labserv:~$
```

( ) Es werden gar keine Mails verschickt.

( ) Es werden zwei Mails verschickt an Prof. Fix, eine davon mit der sinngemäßen Nachricht „unzustellbare Mail, Empfänger nicht vorhanden“.

( ) Nach der Eingabe der Zeile „Hallo Leute!“ hat Prof. Fix Ctrl-D gedrückt.

( ) Hätte er bei Cc: nochmal fix eingegeben, hätte er drei Mails bekommen.

Aufgabe 2

Mit dem nicht in der Vorlesung besprochenen Kommando `comm` *Datei1 Datei2* kann man zwei **sortierte** Dateien zeilenweise vergleichen. Wie kann man sich behelfen, wenn man zwei unsortierte Eingabedateien, sagen wir `ud1` und `ud2`, vorliegen hat.

( ) `comm ‘sort ud1‘ ‘sort ud2‘`

( ) `comm 1<sort ud1 2<sort ud2`

( ) `sort ud1 | sort ud2 | comm`

( ) `(sort ud1)> comm <(sort ud2)`

( ) Geht mit keiner der Lösungen oben.

Aufgabe 3

Mit dem nicht in der Vorlesung besprochenen Kommando `basename` kann man sich den vom Dateipfad und einem ggf. vorhandenen Suffix befreiten „Basisnamen“ ausgeben lassen.

Was liefert der folgende Aufruf, wenn Teilnehmer fix in seinem Heimatverzeichnis ist?

```
fix@labserv:~$ basename `pwd`
```

_____

Aufgabe 4

Mit dem `cal`-Kommando kann man sich schöne Kalenderausgaben erzeugen lassen. Wieso weicht die Zählung für 2011 von der für 2012 ab?

```
fix@labserv:~$ cal 2 2011
    Februar 2011
So Mo Di Mi Do Fr Sa
       1  2  3  4  5
 6  7  8  9 10 11 12
13 14 15 16 17 18 19
20 21 22 23 24 25 26
27 28

fix@labserv:~$ cal 2 2011 | wc -lw
      8      37
fix@labserv:~$ cal 2 2012 | wc -lw
      8      38
fix@labserv:~$
```

Kurze Antwort:

Aufgabe 5

Mit dem `md5sum`-Kommando berechnet man Prüfsummen (Signaturen, Hashwerte) nach dem MD5-Verfahren. Mit der Option `-c` prüft `md5sum`, ob die Angaben in der angegebenen Datei, hier `mdcheck`, gültig sind, d.h. stimmen die dort angegebenen Prüfsummen noch mit den jetzt errechneten Prüfsummen für die genannten Dateien überein. Prof. Fix wollte besonders schlau sein und hat auch die Datei `mdcheck` gehashed. Wieso tritt die Fehlermeldung unten auf?

```
fix@labserv:~$ md5sum liste1
6a2f06060962ba569df187db13a96c6a  liste1
fix@labserv:~$ md5sum liste2
e9e2a835cdbe770feb5446f6769594f4  liste2
fix@labserv:~$ md5sum liste1 >mdcheck
fix@labserv:~$ md5sum liste2 >>mdcheck
fix@labserv:~$ md5sum -c mdcheck
liste1: OK
liste2: OK
fix@labserv:~$ md5sum mdcheck >>mdcheck
fix@labserv:~$ md5sum -c mdcheck
liste1: OK
liste2: OK
mdcheck: FEHLSCHLAG
md5sum: Warnung: 1 von 3 berechneten Prüfsumme passten NICHT
fix@labserv:~$
```

Kurze Antwort:

Aufgabe 6

Das nicht in der Vorlesung behandelte `stat`-Kommando liefert Informationen über den Status einer Datei oder eines Dateisystems. Ergänzen Sie die Ausgabe des Kommandos `ls -ldi`. Hinweis: Jeder Strich entspricht einem nichtleeren Zeichen des Ausgabe.

```
fix@labserv:~$ ls -ldi /home

- ---------- -- ---- ---- ---- --- Jun ----- /home
fix@labserv:~$ stat /home
  File: »/home«
  Size: 4096            Blocks: 8          IO Block: 4096
Verzeichnis
Device: 809h/2057d      Inode: 2           Links: 46
Access: (0755/drwxr-xr-x)  Uid: (   0/   root)  Gid: (   0/
root)
Access: 2011-07-12 19:07:36.000000000 +0200
Modify: 2011-06-15 22:40:29.000000000 +0200
Change: 2011-06-15 22:40:29.000000000 +0200
fix@labserv:~$
```

Aufgabe 7

Was ist richtig in Zusammenhang mit dem Kommando `source` bzw. `.` ?

( ) Beide sind sog. Shell-Spezialkommandos (Shell builtin commands).

( ) Das Kommando `.` darf nur auf versteckte Kommandodateien angewandt werden, z. B. mit `. .profile` wie in der Vorlesung gezeigt.

( ) Die Verwendung von `source` bzw. von `.` kann man umgehen, indem man in der aufgerufenen Kommandodatei alle verwendeten Umgebungsvariablen exportiert (mit `export`-Kommando).

( ) Während der Ausführung ist man im Super-user-Status.

Aufgabe 8

Der `getty`-Prozess ...

( ) ... fragt nach dem Login-Namen und ruft dann den `login`-Prozess auf.

( ) ... wird vom `init`-Prozess gestartet.

( ) ... erbt alle Zombie-Prozesse.

( ) ... startet nach einer Abmeldung des Terminalnutzers automatisch neu.

( ) ... wird sich in der Regel bei erfolgreicher Anmeldung mit der Login-Shell überladen.

Aufgabe 9

Das folgende Beispiel zu Here-Dokumenten fand sich auf der Web-Seite von http://bash.cyberciti.biz/guide/Here_documents

HERE document and mail command

For example, write an email using the mail command. Create a shell script called tapebackup1.sh:

```
#!/bin/bash
# run tar command and dump data to tape
tar -cvf /dev/st0 /www /home 2>/dev/null

# Okay find out if tar was a success or a failure
[ $? -eq 0 ] && status="Success!" || status="Failed!!!"

# write an email to admin
mail -s 'Backup status' vivek@nixcraft.co.in<<END_OF_EMAIL

The backup job finished.

End date: $(date)
Hostname : $(hostname)
Status : $status

END_OF_EMAIL
```

Save and close the file. Run it as follows:

```
chmod +x tapebackup1.sh
./tapebackup1.sh
```

Sample outputs:

```
Subject: Test
From: root <root@www-03.nixcraft.net.in>
Date: 12:57 Am
To: vivek@nixcraft.co.in

The backup job finished.

End date: Thu Sep 17 14:27:35 CDT 2009
Hostname : txvip1.simplyguide.org
Status : Success
```

The script provides the constant multi-line text input to the mail command.

(a) Das Beispiel enthält (mindestens) zwei kleinere Fehler in der gezeigten Ausgabe. Welche?

(b) Was bewirkt `2>/dev/null` im `tar`-Kommando?

(c) Was macht `$(date)` und wie lautet die alternative Syntax in der Vorlesung.

(d) Ersetzen Sie die Konstruktion mit [$? -eq 0] && ... durch ein if-then-else.

(e) Ist es Absicht, dass vor „`<<END_OF_MAIL`“ kein Leerzeichen steht?

(f) Ist es Absicht, dass vor der letzten Zeile mit der Here-Marke `END_OF_MAIL` eine Leerzeile steht?

(g) Warum erfolgt der Aufruf des Skripts aus dem aktuellen Verzeichnis mit `./tapebackup1.sh` statt nur mit `tapebackup1.sh`?

Aufgabe 10

Bei gesetztem SUID-Bit an einem Programm erfolgt das Setzen der neuen effektiven User-Id (EUID) als Nebeneffekt des

( ) `fork`-Systemaufrufs.

( ) `exec`-Systemaufrufs.

( ) `mknod`-Systemaufrufs.

( ) Ladens der Seiten in den virtuellen Speicher.

( ) `chmod`-Kommandos.

Aufgabe 11

Nur eine der folgenden Kommandozeilen wird ohne Fehlermeldung durchlaufen. Welche?

( ) `time date`

( ) `date time`

( ) `time time`

( ) `date date`

Aufgabe 12

In unserer Sammlung gefährlicher und dummer Dateinamen betrachten wir heute den Namen, der aus öffnender und schließender runder Klammer besteht. Kann man ihn erzeugen?

( ) Ja, kann man z. B. mit `touch \(\)` erzeugen.

( ) Ja, kann man z. B. mit `touch "()"` erzeugen.

( ) Ja, kann man sogar ganz locker mit `touch ()` erzeugen.

( ) Ja, kann man mit `touch '()'` erzeugen.

( ) Nein, Klammern sind als Dateiname verboten.

Aufgabe 13

Wenn sich Unix zu einem Teilnehmernamen den zugehörigen Teilnehmer-ID (user id) besorgen muss, dann schaut es in welche Datei, Verzeichnis, Tabelle?

( ) `/home`

( ) `/etc/passwd`

( ) `/etc/shadow`

( ) Prozesstabelle

( ) i-node des ausführenden Programms

Aufgabe 14

Geben Sie den Inhalt der Datei `forum2` an. Beachten Sie das an einer Stelle weggelassene Semikolon („;“) in `forum`!

```
fix@labserv:~$ more forum
In manchen &quot;Foren&quot; sieht
man oft nicht umgesetzte \&quot
Zeichen statt der Anführungszeichen &quot;
fix@labserv:~$ sed -e 's/&quot;/"/g' forum >forum2
fix@labserv:~$
```

Prof. Dr. Lutz Wegner **18. Juli 2011**

Einführung in UNIX
Klausur zum Sommersemester 2011

MUSTERLÖSUNG

Name:.....Vorname:.....

Matr. Nr.:.....Studiengang:.....

Bearbeiten Sie alle Fragen! Bei Ankreuzaufgaben kann mehr als eine Antwort richtig sein. Hilfsmittel sind nicht zugelassen! Falls Sie für die Beantwortung der Aufgaben zusätzlichen Platz benötigen, verwenden Sie die Rückseiten. Bearbeitungszeit 90 Minuten.

Aufgabe	Punkte max.	Punkte erreicht
1	2	
2	2	
3	2	
4	2	
5	2	
6	2	
7	2	
8	2	
9	2+2+2+2+2+2+2	
10	2	
11	2	
12	2	
13	2	
14	2	
Summe	**40**	

Aufgabe 1

Prof. Fix will sich selbst eine Testmail schicken mit dem Betreff „Testmail“, ist sich aber nicht ganz sicher, wo der Betreff hingehört. Was ergibt sich aus dem folgenden Ablauf?

```
fix@labserv:~$ mail fix Testmail
Subject: test
Hallo Leute!
Cc:
fix@labserv:~$
```

( ) Es werden gar keine Mails verschickt.

(X) Es werden zwei Mails verschickt an Prof. Fix, eine davon mit der sinngemäßen Nachricht „unzustellbare Mail, Empfänger nicht vorhanden“.

(X) Nach der Eingabe der Zeile „Hallo Leute!“ hat Prof. Fix Ctrl-D gedrückt.

(X) Hätte er bei Cc: nochmal fix eingegeben, hätte er drei Mails bekommen.

Aufgabe 2

Mit dem nicht in der Vorlesung besprochenen Kommando `comm` *Datei1 Datei2* kann man zwei **sortierte** Dateien zeilenweise vergleichen. Wie kann man sich behelfen, wenn man zwei unsortierte Eingabedateien, sagen wir `ud1` und `ud2`, vorliegen hat.

( ) `comm ‘sort ud1‘ ‘sort ud2‘`

( ) `comm 1<sort ud1 2<sort ud2`

( ) `sort ud1 | sort ud2 | comm`

( ) `(sort ud1)> comm <(sort ud2)`

(X) Geht mit keiner der Lösungen oben.

Aufgabe 3

Mit dem nicht in der Vorlesung besprochenen Kommando `basename` kann man sich den vom Dateipfad und einem ggf. vorhandenen Suffix befreiten „Basisnamen“ ausgeben lassen.

Was liefert der folgende Aufruf, wenn Teilnehmer fix in seinem Heimatverzeichnis ist?

```
fix@labserv:~$ basename `pwd`
```

___ **fix** _____

Aufgabe 4

Mit dem `cal`-Kommando kann man sich schöne Kalenderausgaben erzeugen lassen. Wieso weicht die Zählung für 2011 von der für 2012 ab?

```
fix@labserv:~$ cal 2 2011
    Februar 2011
So Mo Di Mi Do Fr Sa
       1  2  3  4  5
 6  7  8  9 10 11 12
13 14 15 16 17 18 19
20 21 22 23 24 25 26
27 28

fix@labserv:~$ cal 2 2011 | wc -lw
      8      37
fix@labserv:~$ cal 2 2012 | wc -lw
      8      38
fix@labserv:~$
```

Kurze Antwort:

2012 ist Schaltjahr, daher hat der Februar einen Tag (das Wort 29) mehr.

Aufgabe 5

Mit dem `md5sum`-Kommando berechnet man Prüfsummen (Signaturen, Hashwerte) nach dem MD5-Verfahren. Mit der Option `-c` prüft `md5sum`, ob die Angaben in der angegebenen Datei, hier `mdcheck`, gültig sind, d.h. stimmen die dort angegebenen Prüfsummen noch mit den jetzt errechneten Prüfsummen für die genannten Dateien überein. Prof. Fix wollte besonders schlau sein und hat auch die Datei `mdcheck` gehashed. Wieso tritt die Fehlermeldung unten auf?

```
fix@labserv:~$ md5sum liste1
6a2f06060962ba569df187db13a96c6a  liste1
fix@labserv:~$ md5sum liste2
e9e2a835cdbe770feb5446f6769594f4  liste2
fix@labserv:~$ md5sum liste1 >mdcheck
fix@labserv:~$ md5sum liste2 >>mdcheck
fix@labserv:~$ md5sum -c mdcheck
liste1: OK
liste2: OK
fix@labserv:~$ md5sum mdcheck >>mdcheck
fix@labserv:~$ md5sum -c mdcheck
liste1: OK
liste2: OK
mdcheck: FEHLSCHLAG
md5sum: Warnung: 1 von 3 berechneten Prüfsumme passten NICHT
fix@labserv:~$
```

Kurze Antwort:

Zuerst wird für mdcheck die Prüfsumme berechnet, dann diese angehängt an die Datei, wodurch sich aber ein anderer Prüfsummenwert ergibt.

Aufgabe 6

Das nicht in der Vorlesung behandelte `stat`-Kommando liefert Informationen über den Status einer Datei oder eines Dateisystems. Ergänzen Sie die Ausgabe des Kommandos `ls -ldi`. Hinweis: Jeder Strich entspricht einem nichtleeren Zeichen des Ausgabe.

```
fix@labserv:~$ ls -ldi /home
2 drwxr-xr-x 46 root root 4096 15. Jun 22:40 /home
- ---------- -- ---- ---- ---- --- Jun ----- /home
fix@labserv:~$ stat /home
  File: »/home«
  Size: 4096            Blocks: 8          IO Block: 4096
Verzeichnis
Device: 809h/2057d      Inode: 2           Links: 46
Access: (0755/drwxr-xr-x)  Uid: (   0/   root)  Gid: (   0/
root)
Access: 2011-07-12 19:07:36.000000000 +0200
Modify: 2011-06-15 22:40:29.000000000 +0200
Change: 2011-06-15 22:40:29.000000000 +0200
fix@labserv:~$
```

Aufgabe 7

Was ist richtig in Zusammenhang mit dem Kommando `source` bzw. `.` ?

(X) Beide sind sog. Shell-Spezialkommandos (Shell builtin commands).

( ) Das Kommando `.` darf nur auf versteckte Kommandodateien angewandt werden, z. B. mit `. .profile` wie in der Vorlesung gezeigt.

( ) Die Verwendung von `source` bzw. von `.` kann man umgehen, indem man in der aufgerufenen Kommandodatei alle verwendeten Umgebungsvariablen exportiert (mit `export`-Kommando).

( ) Während der Ausführung ist man im Super-user-Status.

Aufgabe 8

Der `getty`-Prozess ...

(X) ... fragt nach dem Login-Namen und ruft dann den `login`-Prozess auf.

(X) ... wird vom `init`-Prozess gestartet.

( ) ... erbt alle Zombie-Prozesse.

(X) ... startet nach einer Abmeldung des Terminalnutzers automatisch neu.

(X) ... wird sich in der Regel bei erfolgreicher Anmeldung mit der Login-Shell überladen.

Aufgabe 9

Das folgende Beispiel zu Here-Dokumenten fand sich auf der Web-Seite von http://bash.cyberciti.biz/guide/Here_documents

HERE document and mail command

For example, write an email using the mail command. Create a shell script called tapebackup1.sh:

```
#!/bin/bash
# run tar command and dump data to tape
tar -cvf /dev/st0 /www /home 2>/dev/null

# Okay find out if tar was a success or a failure
[ $? -eq 0 ] && status="Success!" || status="Failed!!!"

# write an email to admin
mail -s 'Backup status' vivek@nixcraft.co.in<<END_OF_EMAIL

The backup job finished.

End date: $(date)
Hostname : $(hostname)
Status : $status

END_OF_EMAIL
```

Save and close the file. Run it as follows:

```
chmod +x tapebackup1.sh
./tapebackup1.sh
```

Sample outputs:

```
Subject: Test
From: root <root@www-03.nixcraft.net.in>
Date: 12:57 Am
To: vivek@nixcraft.co.in

The backup job finished.

End date: Thu Sep 17 14:27:35 CDT 2009
Hostname : txvip1.simplyguide.org
Status : Success
```

The script provides the constant multi-line text input to the mail command.

(a) Das Beispiel enthält (mindestens) zwei kleinere Fehler in der gezeigten Ausgabe. Welche?

12:57 AM Subject:Backup status (nicht Test) Status : Success!
Versendezeit und Testzeit sind verdächtig verschieden, hostname falsch

(b) Was bewirkt `2>/dev/null` im `tar`-Kommando?

Fehlermeldungen werden unterdrückt

(c) Was macht `$(date)` und wie lautet die alternative Syntax in der Vorlesung.

Kommandosubstitution (das aktuelle Datum tritt an die Stelle des Aufrufs) alternativ 'date'

(d) Ersetzen Sie die Konstruktion mit [$? -eq 0] && ... durch ein if-then-else.

```
if [ $? -eq 0 ]
then staus="Success!"
else status="Failed!!!"
fi
```

(e) Ist es Absicht, dass vor „<<END_OF_MAIL“ kein Leerzeichen steht?

Ein Leerzeichen davor würde nicht schaden (und wäre üblich).

(f) Ist es Absicht, dass vor der letzten Zeile mit der Here-Marke `END_OF_MAIL` eine Leerzeile steht?

Leerzeile könnte entfallen.

(g) Warum erfolgt der Aufruf des Skripts aus dem aktuellen Verzeichnis mit `./tapebackup1.sh` statt nur mit `tapebackup1.sh`?

Aus Sicherheitsgründen nimmt man das Arbeitsverzeichnis nicht im Suchpfad auf.

Aufgabe 10

Bei gesetztem SUID-Bit an einem Programm erfolgt das Setzen der neuen effektiven User-Id (EUID) als Nebeneffekt des

( ) `fork`-Systemaufrufs.

(X) `exec`-Systemaufrufs.

( ) `mknod`-Systemaufrufs.

( ) Ladens der Seiten in den virtuellen Speicher.

( ) `chmod`-Kommandos.

Aufgabe 11

Nur eine der folgenden Kommandozeilen wird ohne Fehlermeldung durchlaufen. Welche?

(X) `time date`

( ) `date time`

( ) `time time`

( ) `date date`

Aufgabe 12

In unserer Sammlung gefährlicher und dummer Dateinamen betrachten wir heute den Namen, der aus öffnender und schließender runder Klammer besteht. Kann man ihn erzeugen?

(X) Ja, kann man z. B. mit `touch \(\)` erzeugen.

(X) Ja, kann man z. B. mit `touch "()"` erzeugen.

( ) Ja, kann man sogar ganz locker mit `touch ()` erzeugen.

(X) Ja, kann man mit `touch '()'` erzeugen.

( ) Nein, Klammern sind als Dateiname verboten.

Aufgabe 13

Wenn sich Unix zu einem Teilnehmernamen den zugehörigen Teilnehmer-ID (user id) besorgen muss, dann schaut es in welche Datei, Verzeichnis, Tabelle?

( ) `/home`

(X) `/etc/passwd`

( ) `/etc/shadow`

( ) Prozesstabelle

( ) i-node des ausführenden Programms

Aufgabe 14

Geben Sie den Inhalt der Datei `forum2` an. Beachten Sie das an einer Stelle weggelassene Semikolon („;“) in `forum`!

```
fix@labserv:~$ more forum
In manchen &quot;Foren&quot; sieht
man oft nicht umgesetzte \&quot
Zeichen statt der Anführungszeichen &quot;
fix@labserv:~$ sed -e 's/&quot;/"/g' forum >forum2
fix@labserv:~$
```

In manchen "Foren" sieht
man oft nicht umgesetzte \&quot
Zeichen statt der Anführungszeichen "

Prof. Dr. Lutz Wegner
Stefan Achler **17. Februar 2012**

Einführung in UNIX
Klausur zum Wintersemester 2011/12

Name:.....Vorname:.....

Matr. Nr.:..... Studiengang:.....

Bearbeiten Sie alle Fragen! Bei Ankreuzaufgaben kann mehr als eine Antwort richtig sein. Ein fehlendes oder falsches Kreuz führt zu einem Punktabzug, negative Punkte sind nicht möglich. Hilfsmittel sind nicht zugelassen! Falls Sie für die Beantwortung der Aufgaben zusätzlichen Platz benötigen, verwenden Sie die Rückseiten.

Aufgabe	Punkte max.	Punkte erreicht
1	2	
2	2	
3	2	
4	2	
5	2	
6	2	
7	2	
8	2	
9	2	
10	2	
11	2	
12	2	
13	2	
14	2	
15	2	
16	12	
Summe	**42**	

Aufgabe 1

Waisenkinder und Zombies werden unter Unix oft verwechselt. Welche der Aussagen zu diesen Prozessen sind richtig?

( ) Init erbt alle Waisenkinder.

( ) Einen Zombieprozess kann man mit `kill -9` nicht „umbringen“, weil er bereits terminiert ist.

( ) Ein Waisenkind ist ein Prozess, der noch kein `wait` gemacht hat.

( ) Ein Zombieprozess ist ein Prozess, der noch kein `exit` gemacht hat.

( ) Ein Zombieprozess wird von `ps` mit dem Status `Z` angezeigt.

( ) Ein Prozess, der ein Waisenkind ist, wird von `ps` mit dem Zustand `O` (für orphan) angezeigt.

Aufgabe 2

Versucht ein Prozess aus `/dev/null` (dem „Biteimer“) zu lesen, dann

( ) blockiert der Prozess so lange, bis aus `/dev/null` Daten angeliefert werden.

( ) erhält der Prozess sofort das Datei-/Eingabeende `EOT` (`CTRL-D`) signalisiert.

( ) bricht der Prozess mit `EXIT 1` ab.

( ) liest der Prozess eine beliebig lange Folge von zufälligen alpha-numerischen Zeichen (ohne Sonderzeichen und ohne Newline).

( ) wird er automatisch zum Vordergrundprozess und `/dev/null` wird durch die Standardeingabe (das eigene Terminal) ersetzt.

Aufgabe 3

Werden in einer Pipeline wie z. B. `cat index | sort | uniq` mehrere Kommandos miteinander verbunden, dann

( ) sind Optionen für die Kommandos nicht möglich, also z. B. kein `sort -n`.

( ) laufen alle Kommandos in der Pipeline unter dem selben Prozessidentifier.

( ) werden mögliche Fehlerausgaben in der Pipeline weitergereicht und der letzte Prozess (oben `uniq`) entscheidet, wo `stderr` erscheint.

( ) stecken implizit Puffer zwischen den Prozessen, die das Schreiben des linken Prozesses (etwa `sort`) mit dem Lesen des rechten (etwa `uniq`) ausgleichen.

( ) werden Zeilenumbrüche (Newlines) automatisch durch Leerzeichen (Blank) ersetzt.

Aufgabe 4

Betrachten Sie das folgende Shellskript `woche`, das sehr ähnlich dem im Kurs gezeigten Programm `uhr` arbeitet. Ergänzen Sie die beiden Lücken!

```
set _____________________
if [ $1 == "Fr" ]
then echo "Das Wochenende lacht!"
elif [ $1 == "Mo" ]
then echo "Die Woche fängt gut an!"
elif [ $1 == "Sa" _____________ $1 == "So" ]
then echo "Samstag und Sonntag bleibt die Mensa zu."
else echo "Di-Mi-Do Professoren arbeiten jetzt."
fi
```

Aufgabe 5

Anders als im Kurs ursprünglich gezeigt, nehmen wir das Arbeitsverzeichnis nicht mehr in den Kommandopfad `$PATH` auf. Warum ist das so und was bedeutet das, etwa für das Shell-Skript `woche` oben?

( ) Sofern wir `woche` nicht in ein anderes Verzeichnis (z. B. ein eigenes `bin`) schieben, das im Pfad steht, erfolgt der Aufruf dann mit `./woche`.

( ) Sofern wir `woche` nicht in ein anderes Verzeichnis (z. B. ein eigenes `bin`) schieben, das im Pfad steht, erfolgt der Aufruf dann mit `./woche:$PATH`.

( ) Man spart sich das Ausführungsrecht an Shell-Skripten, d. h. das übliche `chmod +x woche` entfällt.

( ) Man begegnet so der Gefahr, dass man unwissentlich im Arbeitsverzeichnis ein fremdes und gefährliches Programm speichert und es unbeabsichtigt aufruft, etwa weil es so heisst wie ein gängiges Kommando.

( ) Die Aufname des Arbeitsverzeichnisses in den Kommandopfad `$PATH` ist gar nicht mehr möglich (wurde abgeschafft mit der Änderung bei `chown`).

Aufgabe 6

Wir betrachten zwei ziemlich verrückte here-Dokumente (Shell-Skripte). Welche Ausgaben werden bei Aufruf der jeweiligen Skripte produziert, wenn `ls` zehn nichtversteckte Einträge liefert?

```
cat <<ENDE
`ls | wc -l`
ENDE
```

Ausgabe 1: _____

```
cat <<ENDE
ls | wc -l
ENDE
```

Ausgabe 2: _____

Aufgabe 7

In unserer Sammlung gefährlicher und dummer Dateinamen betrachten wir heute den Namen, der nur aus dem Semikolon „`;`“ besteht. Kann man ihn erzeugen?

( ) Ja, kann man z. B. mit `touch \;` erzeugen.

( ) Ja, kann man z. B. mit `touch ";"` erzeugen.

( ) Ja, kann man sogar ganz locker mit `touch ;` erzeugen.

( ) Ja, kann man mit `touch ';'` erzeugen.

( ) Nein, das Semikolon (der Strichpunkt) ist als Dateiname verboten, weil es als Kommandotrenner gebraucht wird.

Aufgabe 8

Wird mit `ln` *Ziel Linkname* ein (harter) Link für eine schon existierende Datei *Ziel* eingerichtet, dann

( ) erhöht sich der Link-Zähler für *Ziel* (und damit auch für *Linkname*) um eins.

( ) wird für *Linkname* ein neuer i-Knoten (inode) angelegt.

( ) darf *Ziel* nicht selbst schon ein Alias sein (muss also der Originalname der Datei sein).

( ) erscheint bei `ls -l` für *Linkname* als erstes Zeichen der Buchstabe „`v`“ statt dem üblichen „`-`“ für Normaldateien.

( ) zeigt `ls -i` anschließend für *Ziel* und *Linkname* die selbe i-Nummer.

Aufgabe 9

Damit sich zwei oder mehr Prozesse ein Textsegment (den Programmcode) im virtuellen Adressraum teilen können, ...

( ) darf der Programmcode nicht selbstveränderlich sein, was heute sowieso nicht mehr üblich ist.

( ) müssen die Prozesse den selben Besitzer (user-id) haben.

( ) müssen die Prozesse den selben Vaterprozess (parent process id) haben.

( ) dürfen die Prozesse kein `fork()` machen.

( ) muss das SUID-Bit gesetzt sein.

Aufgabe 10

Welche Aussagen sind richtig bezüglich des Kommandos „`.`“ (`source`)?

( ) Dieses Kommando verlangt, dass die auszuführende Datei versteckt ist, also der Dateiname mit „`.`“ anfängt, wie in `.profile`.

( ) Dieses Kommando liest und führt Kommandos in der gegenwärtigen Shell aus.

( ) Dieses Kommando exportiert Variablenwerte aus der Untershell in die aufrufende Obershell.

( ) Dieses Kommando ist shell-intern (ist ein „shell built-in command“).

( ) Dieses Kommando gibt es erst seit neuem, speziell nur für die Nachfolger der Bourne-Shell `sh`.

Aufgabe 11

Die Shell-Variable # gibt die Anzahl der Argumente beim Aufruf des Kommandos an. Was ist richtig?

( ) Die Zählung schließt `$0` ein.

( ) Der Wert verändert sich ggf. als Folge eines `set`-Kommandos.

( ) Der Wert verändert sich ggf. als Folge eines `shift`-Kommandos.

( ) Der Wert wird beim Aufruf festgestellt und ist danach für die Dauer der Abarbeitung unveränderlich.

( ) Der Wert kann maximal „`9`“ sein, weil `$9` der höchste benennbare Positionsparameter ist.

Aufgabe 12

Was ist der Unterschied zwischen `cmp` und `diff`?

( ) Keinerlei Unterschied, sind nur Aliasnamen auf das selbe binary.

( ) `cmp` stoppt beim ersten unterschiedlichen Zeichen.

( ) `diff` stoppt beim ersten unterschiedlichen Zeichen.

( ) `cmp` arbeitet mit einem Hashwert und kann (in sehr unwahrscheinlichen Fällen) die Gleichheit zweier Dateien anzeigen, obwohl die Dateien sich unterscheiden.

( ) `diff` kann nur vorsortierte Dateien vergleichen.

Aufgabe 13

Was gilt für das unter UNIX übliche Runterfahren einer Rechenanlage mit `shutdown`?

( ) Braucht man unter Linux nicht und gibt es auch nicht mehr.

( ) Gibt es auch weiterhin unter Linux und warnt alle angemeldeten Teilnehmer, dass die Anlage abgeschaltet wird.

( ) Gibt es auch weiterhin unter Linux und muss mit einer Zeitangabe versehen werden, wobei `shutdown now` das sofortige Runterfahren bewirkt.

( ) Unter UNIX muss der `init`-Prozess zuletzt vom super-user von Hand per `CTRL-D` gestoppt werden.

( ) `shutdown` sendet den Prozessen das SIGTERM-Signal, das es ihnen erlaubt, geordnet abzubrechen, wenn sie darauf eingerichtet sind.

Aufgabe 14

Teilnehmer `fix` möchte verhindern, dass jemand mit `write` oder `talk` mit ihm online kommuniziert, weil er nicht gestört werden will (Email zu empfangen wäre ok). Was ist richtig?

( ) Schreibzugriff auf das eigene Terminal für Fremde kann man nicht unterbinden.

( ) Mit `mesg n` wird der Schreibzugriff auf sein Terminal gesperrt und Teilnehmer `fix` kann die Einstellung selbst vornehmen.

( ) Mit `mesg -n fix` wird der Schreibzugriff gesperrt für Teilnehmer `fix`, aber nur der super-user kann das einrichten.

( ) Geht durch editieren der Datei `/etc/blacklist`, dort die Zeile `fix::*` eintragen, das sperrt dann allen anderen Teilnehmern den Schreibzugriff auf das Terminal von `fix`. Alternativ die Liste der zu sperrenden Teilnehmer mit Komma getrennt für indivuelle Blockierung. Die Datei `/etc/blacklist` kann `fix` selbst editieren, z. B. mit dem `vi`.

( ) Geht wie in der Alternative oben beschrieben, aber `/etc/blacklist` kann nur der super-user editieren.

Aufgabe 15

Das `read`-Kommando liest von der Standardeingabe und weist die gelesenen Wörter zeilenweise den in der Argumentliste genannten Variablen zu. Was kann man als das Gegenteil von `read` auffassen, d. h. was gibt Argumente auf der Standardausgabe aus?

( ) `write` ( ) `echo` ( ) `cat` ( ) `tee` ( ) `:>`

Aufgabe 16

Betrachten Sie das folgende Shell-Skript `mvupper`.

```
if [ $# != 1 ]
then
  echo "usage: $0 dir"; exit 1
else
  set `ls "$1"`
  for n
  do
    mv "$n" `echo $n | tr "[a-z]" "[A-Z]"`
  done
fi
```

(a) Was bewirkt der Aufruf mit `./mvupper .` in einem Testverzeichnis, das neben `mvupper` nur noch die drei Normaldateien `pi`, `pa` und `po` enthält.

(b) Was hätte der Aufruf mit `./mvupper . ..` bewirkt?

(c) Wäre eine Fehlermeldung erfolgt, wenn im Verzeichnis eine der drei Dateien `pi`, `pa` oder `po` großgeschrieben gewesen wäre, also z. B. `PI` statt `pi`? Wenn ja, wie lautet die Fehlermeldung sinngemäß, wenn nein, warum kommt keine Meldung.

(d) Warum mißlingt ein nochmaliger Aufruf mit `./mvupper .` ?

(e) Was ist zu ändern für ein zweites Shell-Skript `mvlower` mit offensichtlicher Bedeutung?

(f) Was wäre zu ändern, um auch versteckte Dateien mit `mv` in der gezeigten Art umzubenennen?

ENDE DER KLAUSUR

Prof. Dr. Lutz Wegner
Stefan Achler **17. Februar 2012**

Einführung in UNIX
Klausur zum Wintersemester 2011/12

MUSTERLÖSUNG

Name:.....Vorname:.....

Matr. Nr.:..... Studiengang:.....

Bearbeiten Sie alle Fragen! Bei Ankreuzaufgaben kann mehr als eine Antwort richtig sein. Ein fehlendes oder falsches Kreuz führt zu einem Punktabzug, negative Punkte sind nicht möglich. Hilfsmittel sind nicht zugelassen! Falls Sie für die Beantwortung der Aufgaben zusätzlichen Platz benötigen, verwenden Sie die Rückseiten.

Aufgabe	Punkte max.	Punkte erreicht
1	2	
2	2	
3	2	
4	2	
5	2	
6	2	
7	2	
8	2	
9	2	
10	2	
11	2	
12	2	
13	2	
14	2	
15	2	
16	12	
Summe	**42**	

Aufgabe 1

Waisenkinder und Zombies werden unter Unix oft verwechselt. Welche der Aussagen zu diesen Prozessen sind richtig?

(X) Init erbt alle Waisenkinder.

(X) Einen Zombieprozess kann man mit `kill -9` nicht „umbringen“, weil er bereits terminiert ist.

( ) Ein Waisenkind ist ein Prozess, der noch kein `wait` gemacht hat.

( ) Ein Zombieprozess ist ein Prozess, der noch kein `exit` gemacht hat.

(X) Ein Zombieprozess wird von `ps` mit dem Status `Z` angezeigt.

( ) Ein Prozess, der ein Waisenkind ist, wird von `ps` mit dem Zustand `O` (für orphan) angezeigt.

Aufgabe 2

Versucht ein Prozess aus `/dev/null` (dem „Biteimer“) zu lesen, dann

( ) blockiert der Prozess so lange, bis aus `/dev/null` Daten angeliefert werden.

(X) erhält der Prozess sofort das Datei-/Eingabeende `EOT` (`CTRL-D`) signalisiert.

( ) bricht der Prozess mit `EXIT 1` ab.

( ) liest der Prozess eine beliebig lange Folge von zufälligen alpha-numerischen Zeichen (ohne Sonderzeichen und ohne Newline).

( ) wird er automatisch zum Vordergrundprozess und `/dev/null` wird durch die Standardeingabe (das eigene Terminal) ersetzt.

Aufgabe 3

Werden in einer Pipeline wie z. B. `cat index | sort | uniq` mehrere Kommandos miteinander verbunden, dann

( ) sind Optionen für die Kommandos nicht möglich, also z. B. kein `sort -n`.

( ) laufen alle Kommandos in der Pipeline unter dem selben Prozessidentifier.

( ) werden mögliche Fehlerausgaben in der Pipeline weitergereicht und der letzte Prozess (oben `uniq`) entscheidet, wo `stderr` erscheint.

(X) stecken implizit Puffer zwischen den Prozessen, die das Schreiben des linken Prozesses (etwa `sort`) mit dem Lesen des rechten (etwa `uniq`) ausgleichen.

( ) werden Zeilenumbrüche (Newlines) automatisch durch Leerzeichen (Blank) ersetzt.

Aufgabe 4

Betrachten Sie das folgende Shellskript `woche`, das sehr ähnlich dem im Kurs gezeigten Programm `uhr` arbeitet. Ergänzen Sie die beiden Lücken!

```
set ___`date`_____
if [ $1 == "Fr" ]
then echo "Das Wochenende lacht!"
elif [ $1 == "Mo" ]
then echo "Die Woche fängt gut an!"
elif [ $1 == "Sa" ____-o____ $1 == "So" ]
then echo "Samstag und Sonntag bleibt die Mensa zu."
else echo "Di-Mi-Do Professoren arbeiten jetzt."
fi
```

Aufgabe 5

Anders als im Kurs ursprünglich gezeigt, nehmen wir das Arbeitsverzeichnis nicht mehr in den Kommandopfad `$PATH` auf. Warum ist das so und was bedeutet das, etwa für das Shell-Skript `woche` oben?

(X) Sofern wir `woche` nicht in ein anderes Verzeichnis (z. B. ein eigenes `bin`) schieben, das im Pfad steht, erfolgt der Aufruf dann mit `./woche`.

( ) Sofern wir `woche` nicht in ein anderes Verzeichnis (z. B. ein eigenes `bin`) schieben, das im Pfad steht, erfolgt der Aufruf dann mit `./woche:$PATH`.

( ) Man spart sich das Ausführungsrecht an Shell-Skripten, d. h. das übliche `chmod +x woche` entfällt.

(X) Man begegnet so der Gefahr, dass man unwissentlich im Arbeitsverzeichnis ein fremdes und gefährliches Programm speichert und es unbeabsichtigt aufruft, etwa weil es so heisst wie ein gängiges Kommando.

( ) Die Aufname des Arbeitsverzeichnisses in den Kommandopfad `$PATH` ist gar nicht mehr möglich (wurde abgeschafft mit der Änderung bei `chown`).

Aufgabe 6

Wir betrachten zwei ziemlich verrückte here-Dokumente (Shell-Skripte). Welche Ausgaben werden bei Aufruf der jeweiligen Skripte produziert, wenn `ls` zehn nichtversteckte Einträge liefert?

```
cat <<ENDE
`ls | wc -l`
ENDE
```

Ausgabe 1: _____ **10** _____

```
cat <<ENDE
ls | wc -l
ENDE
```

Ausgabe 2: ___ **ls | wc -l** ___

Aufgabe 7

In unserer Sammlung gefährlicher und dummer Dateinamen betrachten wir heute den Namen, der nur aus dem Semikolon „`;`“ besteht. Kann man ihn erzeugen?

(X) Ja, kann man z. B. mit `touch \;` erzeugen.

(X) Ja, kann man z. B. mit `touch ";"` erzeugen.

( ) Ja, kann man sogar ganz locker mit `touch ;` erzeugen.

(X) Ja, kann man mit `touch ';'` erzeugen.

( ) Nein, das Semikolon (der Strichpunkt) ist als Dateiname verboten, weil es als Kommandotrenner gebraucht wird.

Aufgabe 8

Wird mit `ln` *Ziel Linkname* ein (harter) Link für eine schon existierende Datei *Ziel* eingerichtet, dann

(X) erhöht sich der Link-Zähler für *Ziel* (und damit auch für *Linkname*) um eins.

( ) wird für *Linkname* ein neuer i-Knoten (inode) angelegt.

( ) darf *Ziel* nicht selbst schon ein Alias sein (muss also der Originalname der Datei sein).

( ) erscheint bei `ls -l` für *Linkname* als erstes Zeichen der Buchstabe „`v`“ statt dem üblichen „`-`“ für Normaldateien.

(X) zeigt `ls -i` anschließend für *Ziel* und *Linkname* die selbe i-Nummer.

Aufgabe 9

Damit sich zwei oder mehr Prozesse ein Textsegment (den Programmcode) im virtuellen Adressraum teilen können, ...

(X) darf der Programmcode nicht selbstveränderlich sein, was heute sowieso nicht mehr üblich ist.

( ) müssen die Prozesse den selben Besitzer (user-id) haben.

( ) müssen die Prozesse den selben Vaterprozess (parent process id) haben.

( ) dürfen die Prozesse kein `fork()` machen.

( ) muss das SUID-Bit gesetzt sein.

Aufgabe 10

Welche Aussagen sind richtig bezüglich des Kommandos „`.`“ (`source`)?

( ) Dieses Kommando verlangt, dass die auszuführende Datei versteckt ist, also der Dateiname mit „`.`“ anfängt, wie in `.profile`.

(X) Dieses Kommando liest und führt Kommandos in der gegenwärtigen Shell aus.

( ) Dieses Kommando exportiert Variablenwerte aus der Untershell in die aufrufende Obershell.

(X) Dieses Kommando ist shell-intern (ist ein „shell built-in command“).

( ) Dieses Kommando gibt es erst seit neuem, speziell nur für die Nachfolger der Bourne-Shell `sh`.

Aufgabe 11

Die Shell-Variable # gibt die Anzahl der Argumente beim Aufruf des Kommandos an. Was ist richtig?

( ) Die Zählung schließt `$0` ein.

(X) Der Wert verändert sich ggf. als Folge eines `set`-Kommandos.

(X) Der Wert verändert sich ggf. als Folge eines `shift`-Kommandos.

( ) Der Wert wird beim Aufruf festgestellt und ist danach für die Dauer der Abarbeitung unveränderlich.

( ) Der Wert kann maximal „`9`“ sein, weil `$9` der höchste benennbare Positionsparameter ist.

Aufgabe 12

Was ist der Unterschied zwischen `cmp` und `diff`?

( ) Keinerlei Unterschied, sind nur Aliasnamen auf das selbe binary.

(X) `cmp` stoppt beim ersten unterschiedlichen Zeichen.

( ) `diff` stoppt beim ersten unterschiedlichen Zeichen.

( ) `cmp` arbeitet mit einem Hashwert und kann (in sehr unwahrscheinlichen Fällen) die Gleichheit zweier Dateien anzeigen, obwohl die Dateien sich unterscheiden.

( ) `diff` kann nur vorsortierte Dateien vergleichen.

Aufgabe 13

Was gilt für das unter UNIX übliche Runterfahren einer Rechenanlage mit `shutdown`?

( ) Braucht man unter Linux nicht und gibt es auch nicht mehr.

(X) Gibt es auch weiterhin unter Linux und warnt alle angemeldeten Teilnehmer, dass die Anlage abgeschaltet wird.

(X) Gibt es auch weiterhin unter Linux und muss mit einer Zeitangabe versehen werden, wobei `shutdown now` das sofortige Runterfahren bewirkt.

( ) Unter UNIX muss der `init`-Prozess zuletzt vom super-user von Hand per `CTRL-D` gestoppt werden.

(X) `shutdown` sendet den Prozessen das SIGTERM-Signal, das es ihnen erlaubt, geordnet abzubrechen, wenn sie darauf eingerichtet sind.

Auch ohne Kreuz als richtig gewertet, weil technisch gesehen init das SIGTERM-Signal sendet

Aufgabe 14

Teilnehmer `fix` möchte verhindern, dass jemand mit `write` oder `talk` mit ihm online kommuniziert, weil er nicht gestört werden will (Email zu empfangen wäre ok). Was ist richtig?

( ) Schreibzugriff auf das eigene Terminal für Fremde kann man nicht unterbinden.

(X) Mit `mesg n` wird der Schreibzugriff auf sein Terminal gesperrt und Teilnehmer `fix` kann die Einstellung selbst vornehmen.

( ) Mit `mesg -n fix` wird der Schreibzugriff gesperrt für Teilnehmer `fix`, aber nur der super-user kann das einrichten.

( ) Geht durch editieren der Datei `/etc/blacklist`, dort die Zeile `fix::*` eintragen, das sperrt dann allen anderen Teilnehmern den Schreibzugriff auf das Terminal von `fix`. Alternativ die Liste der zu sperrenden Teilnehmer mit Komma getrennt für indivuelle Blockierung. Die Datei `/etc/blacklist` kann `fix` selbst editieren, z. B. mit dem `vi`.

( ) Geht wie in der Alternative oben beschrieben, aber `/etc/blacklist` kann nur der super-user editieren.

Aufgabe 15

Das `read`-Kommando liest von der Standardeingabe und weist die gelesenen Wörter zeilenweise den in der Argumentliste genannten Variablen zu. Was kann man als das Gegenteil von `read` auffassen, d. h. was gibt Argumente auf der Standardausgabe aus?

( ) `write` (X) `echo` ( ) `cat` ( ) `tee` ( ) `:>`

Aufgabe 16

Betrachten Sie das folgende Shell-Skript `mvupper`.

```
if [ $# != 1 ]
then
  echo "usage: $0 dir"; exit 1
else
  set `ls "$1"`
  for n
  do
    mv "$n" `echo $n | tr "[a-z]" "[A-Z]"`
  done
fi
```

a) Was bewirkt der Aufruf mit `./mvupper .` in einem Testverzeichnis, das neben `mvupper` nur noch die drei Normaldateien `pi`, `pa` und `po` enthält.

die vier Dateien mvupper, pi, pa und po werden zu MVUPPER, PI, PA und PO umbenannt.

(b) Was hätte der Aufruf mit `./mvupper . ..` bewirkt?

eine Fehlermeldung „usage: ./mvupper dir" weil nur ein Argument zugelassen ist.

(c) Wäre eine Fehlermeldung erfolgt, wenn im Verzeichnis eine der drei Dateien `pi`, `pa` oder `po` großgeschrieben gewesen wäre, also z. B. `PI` statt `pi`? Wenn ja, wie lautet die Fehlermeldung sinngemäß, wenn nein, warum kommt keine Meldung.

ja, mv hätte gemerkt, dass der neue Name gleich ist dem alten.

(d) Warum mißlingt ein nochmaliger Aufruf mit `./mvupper .` ?

weil mvupper nicht mehr existiert und zu MVUPPER geworden ist.

(e) Was ist zu ändern für ein zweites Shell-Skript `mvlower` mit offensichtlicher Bedeutung?

mv "$n" `echo $n | tr "[A-Z]" "[a-z]"`

(f) Was wäre zu ändern, um auch versteckte Dateien mit `mv` in der gezeigten Art umzubenennen?

set `ls -a "$1"`

ENDE DER KLAUSUR

Prof. Dr. Lutz Wegner
Stefan Achler, M.Sc.

20. Juli 2012

Einführung in UNIX
Klausur zum Sommersemester 2012

Name:.....Vorname:.....

Matr. Nr.:.....Studiengang:.....

Bearbeiten Sie alle Fragen! Bei Ankreuzaufgaben kann mehr als eine Antwort richtig sein. Hilfsmittel sind nicht zugelassen! Falls Sie für die Beantwortung der Aufgaben zusätzlichen Platz benötigen, verwenden Sie die Rückseiten. Separate Blätter sind nicht zugelassen.

Aufgabe	Punkte max.	Punkte erreicht
1	2	
2	2	
3	2	
4	2	
5	2	
6	2	
7	2	
8	2	
9	2	
10	2	
11	2	
12	4	
13	2	
14	2	
15	2	
16	2	
17	4	
Summe	**38**	

Aufgabe 1

Bei den kommerziellen und frei verfügbaren Varianten von Unix und Linux gibt es auch eine Auswahl an Kommandointerpretern (Shells), z. B. `csh`, `ksh`, `sh`, `bash`? Was ist richtig?

( ) Jede Variante von Unix bzw. Linux hat genau eine zugehörige Shell.

( ) Die Varianten von Unix und Linux bieten in der Regel mehrere Shells zur Auswahl an. Welche die Login-Shell ist, bestimmt der Eintrag in der Datei `/etc/passwd`.

( ) Man kann sich zwar für eine andere als die vorinstallierte Shell entscheiden, dann lassen sich aber ggf. nicht alle Unix-Kommandos aufrufen.

( ) Ein Teilnehmer kann zwar verschiedene Shells verwenden, muss sich dazu aber immer ab- und neu anmelden.

( ) Da heute alle Applikationen über eine grafische Oberfläche gestartet werden, gibt es unter Linux keinen Kommandointerpreter (Shell) mehr.

Aufgabe 2

Welche Aussage ist richtig in Zusammenhang mit einer `.profile` (oder ähnlich benannten) Datei im Heimatverzeichnis?

( ) Die Datei ist in der Regel verschlüsselt und direkt ausführbar, d. h. ein *binary*.

( ) Die Datei dient der Vorinitialisierung von Umgebungsvariablen und erledigt andere Aufgaben zu Sitzungsbeginn.

( ) Die Datei wird aus der Shell mittels des eingebauten Kommandos „`.`“ (source) aufgerufen (interpretiert), nicht mittels einer Untershell.

( ) Über die Datei prüft Unix das Passwort des Benutzers und stellt die Gruppenberechtigung her.

( ) Verwendet man die Datei zur Speicherung von Benutzereinstellungen von Sitzung zu Sitzung, dann darf es im Heimatverzeichnis keine weiteren versteckten Dateien geben.

Aufgabe 3

In unserer Sammlung gefährlicher und dummer Dateinamen betrachten wir heute den Namen, der nur aus dem Leerzeichen besteht. Kann man ihn erzeugen?

( ) Ja, z.B. mit `echo dumme Sache >" "`.

( ) Ja, z.B. mit `echo genauso dumme Sache >' '`.

( ) Ja, z.B. mit `touch \ `. Hinweis: Nach dem Escape-Zeichen stehe ein Leerzeichen.

( ) Nein, geht nicht.

Aufgabe 4

Ein Prozess wird zum Zombie-Prozess wenn ...

( ) der Vaterprozess gestorben ist.

( ) jemand den `init`-Prozess abgebrochen hat.

( ) der Prozess terminiert, der Vaterprozess aber noch kein `wait` gemacht hat.

( ) der laufende Prozess mit CTRL-Z abgebrochen wird.

( ) in der Prozesstabelle kein Platz mehr ist.

Aufgabe 5

Welche Aussagen sind zu harten und weichen Links (Dateiverweisen) richtig?

( ) Ein weiterer harter Link auf einen Dateinamen erhöht den Linkzähler der Datei.

( ) Harte Links funktionieren nur innerhalb eines Volumes (einer Partition, eines Dateisystems).

( ) Einen weichen Link kann man mit einem vollständigen oder mit einem relativen Pfadnamen auf eine Datei zeigen lassen.

( ) Auf eine Datei kann es nicht gleichzeitig harte und weiche Links geben.

( ) Wird eine Datei gelöscht, auf die weiche Links zeigen, dann werden auch automatisch alle diese weichen Links gelöscht.

Aufgabe 6

Die Tatsache, dass Unterprozesse die Vorbelegungen ihres Vaterprozesses erben, z.B. die Belegungen für die Dateideskriptoren, ist ein Effekt des ...

( ) `fork`-Systemaufrufs.

( ) `exec`-Systemaufrufs.

( ) `dup`-Systemaufrufs.

( ) `eval`-Systemaufrufs.

( ) SUID-Bits am aufgerufenen Prozess.

Aufgabe 7

Das `pstree`-Kommando ist ein Verwandter des `ps`-Kommandos. Die verkürzte Ausgabe sieht man unten.

```
fix@labserv:~$ pstree -u
init-+-NetworkManager
     |-acpid
        ...
     |-6*[getty]
        ...
     |-sshd-+-sshd---sshd(fix)---bash---pstree
     |      |-sshd---sshd(kai)---sftp-server
     |      `-sshd---sshd(kai)---bash
        ...
     `-wpa_supplicant
fix@labserv:~$
```

Was ist richtig?

( ) Wie im Kurs besprochen, ist `init` Urahn aller Prozesse.

( ) Teilnehmer `fix` (und ein Teilnehmer `kai`) arbeitet mit der `bash`.

( ) Teilnehmer `fix` ist mit `labserv` über einen (bzw. mehrfache) `sshd`-Prozess(e) verbunden.

( ) Sechs `getty`-Prozesse warten auf ein login.

( ) Mit `pstree -u` sieht man auch alle Optionen der Kommandozeilen beim Aufruf der Prozesse.

Aufgabe 8

Die Kommandos `shift`, `read`, `export`, `echo`, `cd` sind in der `bash` sog. shell-interne Kommandos (builtin commands). Welches davon müsste nicht zwingend shell-intern sein?

( ) `shift`

( ) `read`

( ) `export`

( ) `echo`

( ) `cd`

Aufgabe 9

Ergänzen Sie die fehlende Zeile unten!

```
fix@labserv:~$ true
fix@labserv:~$ echo $?
0
fix@labserv:~$ false
fix@labserv:~$ echo $?
_________________
fix@labserv:~$
```

Aufgabe 10

Gehört man mehreren Gruppen an, dann möchte man manchmal beim Arbeiten am Rechner seine Gruppenidentität wechseln. Welche Aussage ist richtig?

( ) Geht mit `chgrp`.

( ) Geht mit `newgrp`.

( ) Geht mit `id`.

( ) Nur `root` kann wechseln.

( ) Gibt es nicht mehr, weil der Wechsel eine Sicherheitslücke darstellt.

Aufgabe 11

Beim Herumexperimentieren mit der Ausgabeumlenkung gibt Teilnehmer `fix` die Zeile

```
fix@labserv:~$ Echo hallo 2>1
```

am Terminal ein. Damit erhält er ...

( ) am Terminal eine Fehlermeldung der Art `Echo: Kommando nicht gefunden`.

( ) am Terminal die Ausgabe `hallo`.

( ) am Terminal die Ausgabe `hallo` **und** eine Fehlermeldung der Art `Echo: Kommando nicht gefunden.`

( ) eine Datei mit Namen `1`, die als Inhalt eine Fehlermeldung der Art `Echo: Kommando nicht gefunden` enthält.

( ) am Terminal eine Fehlermeldung der Art `Echo: Kommando nicht gefunden` und eine leere Datei mit Namen `1`.

Aufgabe 12

Die Datei `doppler` enthalte die folgenden Daten:

```
aber aber hallo hallo
was
isn isn isn das das?
```

Beachten Sie das Fragezeichen am letzten Wort. Das Shellskript `exterminator` habe den folgenden Inhalt.

```
while true
do
  read eingabe
  if [ "$eingabe" == "" ]
  then break
  fi
  set $eingabe
  ausgabe=""
  while [ $# -gt 1 ]
  do
    if [ "$1" == "$2" ]
    then shift
    else ausgabe=$ausgabe" "$1; shift
    fi
  done
  echo $ausgabe $1
done
```

Wie lautet die Ausgabe beim Aufruf von `./exterminator <doppler` ? Antwort bitte auf der Rückseite.

Aufgabe 13

Das `date`-Kommando liefert mit der Option `-r` *Dateiname* die Modifikationszeit der Datei, das sleep-Kommando setzt die Ausführung eines Programms für die angegebene Anzahl an Sekunden aus. Man betrachte die folgende Kommandozeile und die beiden Ausgabezeilen (mit Lücke).

```
fix@labserv:~$ date; sleep 5; touch test; date -r test

Do 12. Jul 15:27:50 CEST 2012

_____________________________________________________

fix@labserv:~$
```

Ergänzen Sie die Lücke!

Aufgabe 14

Im gegenwärtigen Verzeichnis `testdir` befinden sich nur die Dateien `pi`, `pa` und `po`. Welche der Alternativen liefern genau die folgende Ausgabe, wenn man sie in die Lücke einsetzt?

```
fix@labserv:~/testdir$ _____________________________

pa  pi  po

fix@labserv:~/testdir$
```

( ) `echo *`

( ) `ls`

( ) `echo -n ‘ls‘`

( ) `echo ‘echo *‘`

( ) `echo ‘ls -nl‘`

Aufgabe 15

Nehmen wir an, Sie haben eine Datei mit dem `crypt`-Kommando verschlüsselt, die Klartext-Datei unwiederbringbar mit `shred` gelöscht und dann dummerweise den Schlüssel (die *passphrase*) vergessen. Wir kommt man an den Klartext heran?

( ) Den super-user (`root`) bitten, die Datei zu entschlüsseln.

( ) Stehen die Aufrufe des `crypt`-Kommandos und des `shred`-Kommandos für das Löschen des Klartexts noch in der HISTORY-Datei, dann können Sie beide mit dem `undo`-Komamando rückgängig machen.

( ) Der Schlüssel wird am Anfang der Datei mitgespeichert (die sog. *magic bytes*). Der Systemaufruf `magic(`*Dateipfad*`)` liefert ihn im Klartext.

( ) Sieht schlecht aus, außer Ihnen fällt der Schlüssel wieder ein! Im Klartext: es gibt keine praktisch anwendbare Methode.

( ) Ein unwiederbringbares Löschen gibt es unter Linux nicht. Alle gelöschten Dateien befinden sich bis zum Herunterfahren in dem Verzeichnis `.wastebasket`.

Aufgabe 16

Die Standardvorbelegung für neu angelegte Verzeichnisse ist 755. Damit können Teilnehmer, die nicht der Besitzer oder root sind, ...

( ) keine neuen Einträge vornehmen und keine bestehende löschen oder umbenennen.

( ) im Verzeichnis lesen.

( ) durch das Verzeichnis navigieren, also in Unterverzeichnisse wechseln.

( ) weder im Verzeichnis lesen noch durch es navigieren in Unterverzeichnisse.

( ) zwar im Verzeichnis lesen, aber nicht in Unterverzeichnisse navigieren.

Aufgabe 17

Das in der Vorlesung nicht besprochene `stat`-Kommando liefert Angaben zum Datei- oder Dateisystemstatus. Ergänzen Sie die Lücken bei folgender Ausgabe!

```
fix@labserv:~/testdir$ ls -li pi
4997872 -rw-r--r-- 1 fix fix 5 12. Jul 16:48 pi
fix@labserv:~/testdir$ stat pi
  File: »pi«
  Size: _______     Blocks: 8          IO Block: 4096
reguläre Datei
Device: 809h/2057d      Inode: _____________ Links: _____
Access: (0____/-_____________)  Uid: ( 1006/ _____________)
Gid: ( 1009/ ____________)
Access: 2011-12-09 11:03:49.000000000 +0100
Modify: 2012-07-12 _____:20.000000000 +0200
Change: 2012-07-12 16:48:20.000000000 +0200
fix@labserv:~/testdir$
```

ENDE DER KLAUSUR

Prof. Dr. Lutz Wegner
Stefan Achler, M.Sc.

20. Juli 2012

Einführung in UNIX
Klausur zum Sommersemester 2012

MUSTERLÖSUNG

Name:.....Vorname:.....

Matr. Nr.:.....Studiengang:.....

Bearbeiten Sie alle Fragen! Bei Ankreuzaufgaben kann mehr als eine Antwort richtig sein. Hilfsmittel sind nicht zugelassen! Falls Sie für die Beantwortung der Aufgaben zusätzlichen Platz benötigen, verwenden Sie die Rückseiten. Separate Blätter sind nicht zugelassen.

Aufgabe	Punkte max.	Punkte erreicht
1	2	
2	2	
3	2	
4	2	
5	2	
6	2	
7	2	
8	2	
9	2	
10	2	
11	2	
12	4	
13	2	
14	2	
15	2	
16	2	
17	4	
Summe	**38**	

Aufgabe 1

Bei den kommerziellen und frei verfügbaren Varianten von Unix und Linux gibt es auch eine Auswahl an Kommandointerpretern (Shells), z. B. `csh`, `ksh`, `sh`, `bash`? Was ist richtig?

( ) Jede Variante von Unix bzw. Linux hat genau eine zugehörige Shell.

(X) Die Varianten von Unix und Linux bieten in der Regel mehrere Shells zur Auswahl an. Welche die Login-Shell ist, bestimmt der Eintrag in der Datei `/etc/passwd`.

( ) Man kann sich zwar für eine andere als die vorinstallierte Shell entscheiden, dann lassen sich aber ggf. nicht alle Unix-Kommandos aufrufen.

( ) Ein Teilnehmer kann zwar verschiedene Shells verwenden, muss sich dazu aber immer ab- und neu anmelden.

( ) Da heute alle Applikationen über eine grafische Oberfläche gestartet werden, gibt es unter Linux keinen Kommandointerpreter (Shell) mehr.

Aufgabe 2

Welche Aussage ist richtig in Zusammenhang mit einer `.profile` (oder ähnlich benannten) Datei im Heimatverzeichnis?

( ) Die Datei ist in der Regel verschlüsselt und direkt ausführbar, d. h. ein *binary*.

(X) Die Datei dient der Vorinitialisierung von Umgebungsvariablen und erledigt andere Aufgaben zu Sitzungsbeginn.

(X) Die Datei wird aus der Shell mittels des eingebauten Kommandos „`.`“ (source) aufgerufen (interpretiert), nicht mittels einer Untershell.

( ) Über die Datei prüft Unix das Passwort des Benutzers und stellt die Gruppenberechtigung her.

( ) Verwendet man die Datei zur Speicherung von Benutzereinstellungen von Sitzung zu Sitzung, dann darf es im Heimatverzeichnis keine weiteren versteckten Dateien geben.

Aufgabe 3

In unserer Sammlung gefährlicher und dummer Dateinamen betrachten wir heute den Namen, der nur aus dem Leerzeichen besteht. Kann man ihn erzeugen?

(X) Ja, z.B. mit `echo dumme Sache >" "`.

(X) Ja, z.B. mit `echo genauso dumme Sache >' '`.

(X) Ja, z.B. mit `touch \ `. Hinweis: Nach dem Escape-Zeichen stehe ein Leerzeichen.

( ) Nein, geht nicht.

Aufgabe 4

Ein Prozess wird zum Zombie-Prozess wenn ...

( ) der Vaterprozess gestorben ist.

( ) jemand den `init`-Prozess abgebrochen hat.

(X) der Prozess terminiert, der Vaterprozess aber noch kein `wait` gemacht hat.

( ) der laufende Prozess mit CTRL-Z abgebrochen wird.

( ) in der Prozesstabelle kein Platz mehr ist.

Aufgabe 5

Welche Aussagen sind zu harten und weichen Links (Dateiverweisen) richtig?

(X) Ein weiterer harter Link auf einen Dateinamen erhöht den Linkzähler der Datei.

(X) Harte Links funktionieren nur innerhalb eines Volumes (einer Partition, eines Dateisystems).

(X) Einen weichen Link kann man mit einem vollständigen oder mit einem relativen Pfadnamen auf eine Datei zeigen lassen.

( ) Auf eine Datei kann es nicht gleichzeitig harte und weiche Links geben.

( ) Wird eine Datei gelöscht, auf die weiche Links zeigen, dann werden auch automatisch alle diese weichen Links gelöscht.

Aufgabe 6

Die Tatsache, dass Unterprozesse die Vorbelegungen ihres Vaterprozesses erben, z.B. die Belegungen für die Dateideskriptoren, ist ein Effekt des ...

(X) `fork`-Systemaufrufs.

( ) `exec`-Systemaufrufs.

( ) `dup`-Systemaufrufs.

( ) `eval`-Systemaufrufs.

( ) SUID-Bits am aufgerufenen Prozess.

Aufgabe 7

Das `pstree`-Kommando ist ein Verwandter des `ps`-Kommandos. Die verkürzte Ausgabe sieht man unten.

```
fix@labserv:~$ pstree -u
init-+-NetworkManager
     |-acpid
        ...
     |-6*[getty]
        ...
     |-sshd-+-sshd---sshd(fix)---bash---pstree
     |      |-sshd---sshd(kai)---sftp-server
     |      `-sshd---sshd(kai)---bash
        ...
     `-wpa_supplicant
fix@labserv:~$
```

Was ist richtig?

(X) Wie im Kurs besprochen, ist `init` Urahn aller Prozesse.

(X) Teilnehmer `fix` (und ein Teilnehmer `kai`) arbeitet mit der `bash`.

(X) Teilnehmer `fix` ist mit `labserv` über einen (bzw. mehrfache) `sshd`-Prozess(e) verbunden.

(X) Sechs `getty`-Prozesse warten auf ein login.

( ) Mit `pstree -u` sieht man auch alle Optionen der Kommandozeilen beim Aufruf der Prozesse.

Aufgabe 8

Die Kommandos `shift`, `read`, `export`, `echo`, `cd` sind in der `bash` sog. shell-interne Kommandos (builtin commands). Welches davon müsste nicht zwingend shell-intern sein?

( ) `shift`

( ) `read`

( ) `export`

(X) `echo`

( ) `cd`

Aufgabe 9

Ergänzen Sie die fehlende Zeile unten!

```
fix@labserv:~$ true
fix@labserv:~$ echo $?
0
fix@labserv:~$ false
fix@labserv:~$ echo $?
1 _________________
fix@labserv:~$
```

Aufgabe 10

Gehört man mehreren Gruppen an, dann möchte man manchmal beim Arbeiten am Rechner seine Gruppenidentität wechseln. Welche Aussage ist richtig?

( ) Geht mit `chgrp`.

(X) Geht mit `newgrp`.

( ) Geht mit `id`.

( ) Nur `root` kann wechseln.

( ) Gibt es nicht mehr, weil der Wechsel eine Sicherheitslücke darstellt.

Aufgabe 11

Beim Herumexperimentieren mit der Ausgabeumlenkung gibt Teilnehmer `fix` die Zeile

```
fix@labserv:~$ Echo hallo 2>1
```

am Terminal ein. Damit erhält er ...

( ) am Terminal eine Fehlermeldung der Art `Echo: Kommando nicht gefunden`.

( ) am Terminal die Ausgabe `hallo`.

( ) am Terminal die Ausgabe `hallo` **und** eine Fehlermeldung der Art `Echo: Kommando nicht gefunden.`

(X) eine Datei mit Namen `1`, die als Inhalt eine Fehlermeldung der Art `Echo: Kommando nicht gefunden` enthält.

( ) am Terminal eine Fehlermeldung der Art `Echo: Kommando nicht gefunden` und eine leere Datei mit Namen `1`.

Aufgabe 12

Die Datei `doppler` enthalte die folgenden Daten:

```
aber aber hallo hallo
was
isn isn isn das das?
```

Beachten Sie das Fragezeichen am letzten Wort. Das Shellskript `exterminator` habe den folgenden Inhalt.

```
while true
do
  read eingabe
  if [ "$eingabe" == "" ]
  then break
  fi
  set $eingabe
  ausgabe=""
  while [ $# -gt 1 ]
  do
    if [ "$1" == "$2" ]
    then shift
    else ausgabe=$ausgabe" "$1; shift
    fi
  done
  echo $ausgabe $1
done
```

Lösung:

aber hallo

was

isn das das?

Wie lautet die Ausgabe beim Aufruf von `./exterminator <doppler` ? Antwort bitte auf der Rückseite.

Aufgabe 13

Das `date`-Kommando liefert mit der Option `-r` *Dateiname* die Modifikationszeit der Datei, das sleep-Kommando setzt die Ausführung eines Programms für die angegebene Anzahl an Sekunden aus. Man betrachte die folgende Kommandozeile und die beiden Ausgabezeilen (mit Lücke).

```
fix@labserv:~$ date; sleep 5; touch test; date -r test

Do 12. Jul 15:27:50 CEST 2012

Do 12. Jul 15:27:55 CEST 2012_________________

fix@labserv:~$
```

Ergänzen Sie die Lücke!

Aufgabe 14

Im gegenwärtigen Verzeichnis `testdir` befinden sich nur die Dateien `pi`, `pa` und `po`. Welche der Alternativen liefern genau die folgende Ausgabe, wenn man sie in die Lücke einsetzt?

```
fix@labserv:~/testdir$ _____________________________

pa  pi  po

fix@labserv:~/testdir$
```

(X) `echo *`

(X) `ls`

( ) `echo -n ‘ls‘`

(X) `echo ‘echo *‘`

( ) `echo ‘ls -nl‘`

Aufgabe 15

Nehmen wir an, Sie haben eine Datei mit dem `crypt`-Kommando verschlüsselt, die Klartext-Datei unwiederbringbar mit `shred` gelöscht und dann dummerweise den Schlüssel (die *passphrase*) vergessen. Wir kommt man an den Klartext heran?

( ) Den super-user (`root`) bitten, die Datei zu entschlüsseln.

( ) Stehen die Aufrufe des `crypt`-Kommandos und des `shred`-Kommandos für das Löschen des Klartexts noch in der HISTORY-Datei, dann können Sie beide mit dem `undo`-Komamando rückgängig machen.

( ) Der Schlüssel wird am Anfang der Datei mitgespeichert (die sog. *magic bytes*). Der Systemaufruf `magic(`*Dateipfad*`)` liefert ihn im Klartext.

(X) Sieht schlecht aus, außer Ihnen fällt der Schlüssel wieder ein! Im Klartext: es gibt keine praktisch anwendbare Methode.

( ) Ein unwiederbringbares Löschen gibt es unter Linux nicht. Alle gelöschten Dateien befinden sich bis zum Herunterfahren in dem Verzeichnis `.wastebasket`.

Aufgabe 16

Die Standardvorbelegung für neu angelegte Verzeichnisse ist 755. Damit können Teilnehmer, die nicht der Besitzer oder root sind, ...

(X) keine neuen Einträge vornehmen und keine bestehende löschen oder umbenennen.

(X) im Verzeichnis lesen.

(X) durch das Verzeichnis navigieren, also in Unterverzeichnisse wechseln.

( ) weder im Verzeichnis lesen noch durch es navigieren in Unterverzeichnisse.

( ) zwar im Verzeichnis lesen, aber nicht in Unterverzeichnisse navigieren.

Aufgabe 17

Das in der Vorlesung nicht besprochene `stat`-Kommando liefert Angaben zum Datei- oder Dateisystemstatus. Ergänzen Sie die Lücken bei folgender Ausgabe!

```
fix@labserv:~/testdir$ ls -li pi
4997872 -rw-r--r-- 1 fix fix 5 12. Jul 16:48 pi
fix@labserv:~/testdir$ stat pi
  File: »pi«
  Size: __5___      Blocks: 8          IO Block: 4096
reguläre Datei
Device: 809h/2057d      Inode: __4997872__ Links: __1__
Access: (0644/-rw-r--r--)  Uid: ( 1006/ ___fix____)
Gid: ( 1009/ ___fix___)
Access: 2011-12-09 11:03:49.000000000 +0100
Modify: 2012-07-12 16:48:20.000000000 +0200
Change: 2012-07-12 16:48:20.000000000 +0200
fix@labserv:~/testdir$
```

ENDE DER KLAUSUR

Stefan Achler M.Sc.
Prof. Dr. Lutz Wegner **18. Februar 2013**

Einführung in UNIX
Klausur zum Wintersemester 2012/13

Name:.....Vorname:.....

Matr. Nr.:..... Studiengang:.....

Bearbeiten Sie alle Fragen! Bei Ankreuzaufgaben kann mehr als eine Antwort richtig sein. Ein fehlendes oder falsches Kreuz führt zu einem Punktabzug, negative Punkte sind nicht möglich. Hilfsmittel sind nicht zugelassen! Falls Sie für die Beantwortung der Aufgaben zusätzlichen Platz benötigen, verwenden Sie die Rückseiten.

Aufgabe	Punkte max.	Punkte erreicht
1	2	
2	2	
3	2	
4	2	
5	2	
6	2	
7	2	
8	2	
9	2	
10	2	
11	2	
12	2	
13	2	
14	2	
15	2	
16	2	
17	2	
18	2	
19	2	
20	2	
21	2	
Summe	**42**	

Aufgabe 1

Wie lautet der Name der von Microsoft ursprünglich für den Intel-8086 vorgenommenen Portierung von AT&T Unix ab 1980?

( ) MS Plan9

( ) Minix

( ) MS Xenix

( ) MS Linux

( ) Open California

Aufgabe 2

Die technischen Möglichkeiten in den siebziger Jahren waren beschränkt. Worauf konnte für die ersten Unix-Entwicklungen, sagen wir bis System V7 1979, verzichtet werden?

( ) Dateisystem

( ) graphische Oberfläche

( ) Shell

( ) Mehrbenutzerbetrieb

( ) eine permanente Internet-Verbindung mit AT&T zum Booten

Aufgabe 3

Was bewirkt `kill -9`, wenn ein Anwender damit einen seiner Prozesse „umbringt“?

( ) Dieser Prozess wird zum Waisenkind.

( ) Dieser Prozess wird beendet (kann nicht abgefangen werden).

( ) Dieser Prozess wird **immer** ein Zombieprozess.

( ) Dieses Kommando bewirkt, dass auch die Shell, in der es eingegeben wurde, terminiert wird.

( ) Allen noch geöffneten Dateien wird das SIGKILL-Signal geschickt.

Aufgabe 4

Im Dateisystem werden unter `/dev` diverse Geräte aufgeführt, z. B. auch `/dev/mem` für den Hauptspeicher. Was gilt für solche Geräte?

( ) Sie haben keinen Besitzer.

( ) Es sind keine Lese-/Schreib-/Ausführungsrechte setzbar.

( ) Alle in `/dev` aufgeführten Geräte müssen vor der ersten Nutzung gemountet werden.

( ) Sie werden über Gerätetreiber angesprochen, sofern es sich nicht um so einfache Pseudogeräte wie `/dev/null` handelt.

( ) Die in `/dev` aufgeführten Namen, z. B. `mem`, `hd`, `lp`, ..., sind geschützte Bezeichner und dürfen nicht für eigene Dateien verwendet werden.

Aufgabe 5

In unserer Sammlung gefährlicher und dummer Dateinamen betrachten wir heute den Namen, der nur aus dem sonst für die Umlenkung genutzten Zeichen „>“ besteht. Kann man ihn erzeugen?

( ) Ja, kann man z. B. mit `touch \>` erzeugen.

( ) Ja, kann man z. B. mit `touch ">"` erzeugen.

( ) Ja, kann man sogar ganz locker mit `touch >` erzeugen.

( ) Ja, kann man mit `touch '>'` erzeugen.

( ) Nein, das Größerzeichen („>“) ist als Dateiname nicht erzeugbar und verboten, weil es für die Umlenkung in der Shell gebraucht wird.

Aufgabe 6

Kann ein User einen Prozess mit der Prozess-ID 1 erzeugen?

( ) Ja, ist möglich und kann von dem User erzwungen werden.

( ) Ja, ist möglich, wenn dieser noch nicht benutzt wird.

( ) Nein, ist nicht möglich. (`init` hat immer die PID 1)

( ) Ja, aber nur als Super-User.

( ) Es gibt unter Unix/Linux keine Prozess-IDs mehr.

Aufgabe 7

Benutzer `fix` möchte die `datei1` in `datei2` mittels `mv datei1 datei2` umbenennen. Welche Kommandofolgen haben den gleichen Effekt?

( ) `cp datei1 datei2; rm datei1`

( ) `cp datei1 datei2 && rm datei1`

( ) `echo -n >datei1; echo -n >datei2`

( ) `ln datei1 datei2; rm datei1`

( ) `mv datei1 .`

Aufgabe 8

Fügen Sie der Variablen `PATH` das `HOME`-Verzeichnis hinzu!

( ) `PATH=$PATH:$HOME`

( ) `$PATH=$PATH:$HOME`

( ) `PATH+=$HOME`

( ) `PATH=$PATH;$HOME`

( ) Geht grundsätzlich nicht, das `HOME`-Verzeichnis lässt sich nicht zu `PATH` hinzufügen.

Aufgabe 9

Eine Datei löscht man mit dem `rm`-Kommando.

( ) Ja.

( ) Nur wenn die Datei nicht versteckt ist.

( ) Nur wenn die Datei schon leer ist, sonst `rm -r`.

( ) Kommt auf die Größe der Datei an.

( ) Nein, geht mit dem `rmdir`-Kommando.

Aufgabe 10

Was wird mit dem `set`-Kommando ohne Argumente angezeigt?

( ) Nur die für den Export freigegebenen Variablen.

( ) Nur die für den Import freigegebenen Variablen.

( ) Alle in der Shell bekannten Variablen.

( ) Nur die read-only Variablen.

( ) Nur die Optionen, mit denen die gegenwärtige Shell aufgerufen wurde.

Aufgabe 11

Welche Aussagen sind für die Kommandoverknüpfung *Kommando1* `||` *Kommando2* richtig?

( ) Beide Kommandos werden in jedem Fall ausgeführt.

( ) *Kommando2* wird nur ausgeführt, wenn *Kommando1* fehlschlägt.

( ) *Kommando2* wird nur ausgeführt, wenn *Kommando1* erfolgreich ausgeführt wird.

( ) *Kommando2* wird nur ausgeführt, wenn *Kommando1* eine nicht-leere Zeichenfolge in die Pipe schickt, unabhängig vom Exit-Status, den *Kommando1* zurückliefert.

( ) Ersetzt die Kommandoverknüpfung *Kommando1*`;` *Kommando2* mit gleicher Wirkung.

Aufgabe 12

Welche Kommandozeilen erzeugen dieselbe Ausgabe wie der Aufruf von `date`?

( ) `echo date`

( ) `echo ‘date‘`

( ) `date | cat`

( ) `cat ‘date‘`

( ) `cat date`

Aufgabe 13

Die Dateirechte am Shell-Skript `here` aus Aufgabe 14 seien 644. Wie lautet das `chmod`-Kommando, damit Besitzer `fix` es ausführen kann?

Aufgabe 14

Gegeben sei das folgende Shell-Skript `here`

```
sh <<ENDE
ls -l
ENDE
```

Welche Ausgabe erzeugt der Aufruf von `./here` ?

( ) Die Zeile `ls -l`.

( ) Eine Auflistung der Dateien im gegenwärtigen Verzeichnis wie bei `ls -l`.

( ) Eine Fehlermeldung `Datei ls nicht gefunden`.

( ) Eine Fehlermeldung `Datei ENDE nicht gefunden`.

( ) Die Dateieigenschaften für die Datei `ENDE` im Langformat.

Aufgabe 15

Gegeben sei das folgende Shell-Skript `zensur`.

```
for word do
    if [ $word = $1 ]
    then
        echo "*****"
    else
        echo $word
    fi
done
```

Was liefert der Aufruf mit `./zensur hallo aber hallo aber ja bitte` ?

Aufgabe 16

Betrachten Sie die folgende Kommandoabfolge des Benutzers `fix`.

```
fix@labserv:~$ id
uid=1006(fix) gid=1009(fix) Gruppen=1009(fix),100(users)
fix@labserv:~$ echo $$
23029
fix@labserv:~$ newgrp users
fix@labserv:~$ id
uid=1006(fix) gid=100(users) Gruppen=1009(fix),100(users)
fix@labserv:~$ echo $$
26801
fix@labserv:~$ exit
fix@labserv:~$ id
uid=1006(fix) gid=1009(fix) Gruppen=1009(fix),100(users)
```

Was kann daraus geschlossen werden?

( ) Der Benutzer `fix` ändert temporär seine Gruppenkennung zu `users`.

( ) Es wird eine neue Untershell erzeugt.

( ) Der Gruppenwechsel erfolgt in der alten Shell.

( ) Um in die alte Gruppenidentität zurückzuwechseln, genügt ein CTRL-D (`exit`).

( ) Um in die alte Gruppenidentität zurückzuwechseln, muss `oldgrp fix` aufgerufen werden.

Aufgabe 17

Betrachten Sie die folgende Kommandofolge. Ergänzen Sie die fehlende Zeile!

```
fix@labserv:~$ ps
PID TTY          TIME CMD
23029 pts/6    00:00:00 bash
29213 pts/6    00:00:00 ps
fix@labserv:~$ bash
fix@labserv:~$ ps
PID TTY          TIME CMD
23029 pts/6    00:00:00 bash
29217 pts/6    00:00:00 bash
29220 pts/6    00:00:00 ps
fix@labserv:~$ echo $$ $#
```

_____

Aufgabe 18

Was erzeugt der Systemaufruf `fork()`, wenn er in einem Prozess *P* aufgerufen wird?

( ) Einen neuen, zu *P* identischen Prozess mit neuer Prozess-ID.

( ) Einen neuen Prozess *Q*, der *P* überlagert.

( ) Ein neues Textsegment mit gleicher Prozess-ID.

( ) Eine leere Prozess-ID ohne Text- und Datensegment (Pseudo-Zombie).

( ) Eine Änderung der User-Bits an *P* (SUID-Bit wird auf 1 gesetzt).

Aufgabe 19

Wir betrachten die Pipeline `ls -i | sort`. Was gilt bezüglich der Ausgabe?

( ) Das `ls`-Kommando wird im interaktiven Modus gestartet und verlangt die Benutzerfreigabe für jeden Eintrag.

( ) Dateinamen mit gleicher i-Nummer (aus `ln`) stehen hintereinander.

( ) Die Zeilen der Ausgabe sind aufsteigend nach i-Nummer angeordnet.

( ) Die Ausgabe erfolgt aufsteigend nach Dateinamen.

( ) Enthält auch die versteckten Dateien „`.`“ und „`..`“.

Aufgabe 20

Welche der folgenden Angaben zu einer Datei werden in den Verzeichnissen (nicht im i-node) selbst abgespeichert?

( ) Der Name der Datei.

( ) Die i-Nummer der Datei (Index des i-node).

( ) Der Besitzer.

( ) Die Dateirechte.

( ) Die Startadressen der Datenblöcke.

Aufgabe 21

Betrachten Sie das Shell-Script `append`. Bei Aufruf mit nur einem Dateinamen als Parameter soll die Eingabe aus `stdin` an die genannte *Zieldatei* angehängt werden. Bei Aufruf mit zwei Dateinamen soll der Inhalt von *Quelldatei* an *Zieldatei* angehangen werden. Füllen Sie die Lücken!

```
if [ _____ -eq ______ ]
then cat _________ $1
else  if [ _____________ ]
    then cat  >> $2 < $1
    else echo "Anleitung: append [Quelldatei] Zieldatei"
    fi
fi
```

ENDE DER KLAUSUR

Stefan Achler M.Sc.
Prof. Dr. Lutz Wegner **18. Februar 2013**

Einführung in UNIX
Klausur zum Wintersemester 2012/13

MUSTERLÖSUNG

Name:..... Vorname:.....

Matr. Nr.:..... Studiengang:.....

Bearbeiten Sie alle Fragen! Bei Ankreuzaufgaben kann mehr als eine Antwort richtig sein. Ein fehlendes oder falsches Kreuz führt zu einem Punktabzug, negative Punkte sind nicht möglich. Hilfsmittel sind nicht zugelassen! Falls Sie für die Beantwortung der Aufgaben zusätzlichen Platz benötigen, verwenden Sie die Rückseiten.

Aufgabe	Punkte max.	Punkte erreicht
1	2	
2	2	
3	2	
4	2	
5	2	
6	2	
7	2	
8	2	
9	2	
10	2	
11	2	
12	2	
13	2	
14	2	
15	2	
16	2	
17	2	
18	2	
19	2	
20	2	
21	2	
Summe	**42**	

Aufgabe 1

Wie lautet der Name der von Microsoft ursprünglich für den Intel-8086 vorgenommenen Portierung von AT&T Unix ab 1980?

( ) MS Plan9

( ) Minix

(X) MS Xenix

( ) MS Linux

( ) Open California

Aufgabe 2

Die technischen Möglichkeiten in den siebziger Jahren waren beschränkt. Worauf konnte für die ersten Unix-Entwicklungen, sagen wir bis System V7 1979, verzichtet werden?

( ) Dateisystem

(X) graphische Oberfläche

( ) Shell

( ) Mehrbenutzerbetrieb

(X) eine permanente Internet-Verbindung mit AT&T zum Booten

Aufgabe 3

Was bewirkt `kill -9`, wenn ein Anwender damit einen seiner Prozesse „umbringt“?

( ) Dieser Prozess wird zum Waisenkind.

(X) Dieser Prozess wird beendet (kann nicht abgefangen werden).

( ) Dieser Prozess wird **immer** ein Zombieprozess.

( ) Dieses Kommando bewirkt, dass auch die Shell, in der es eingegeben wurde, terminiert wird.

( ) Allen noch geöffneten Dateien wird das SIGKILL-Signal geschickt.

Aufgabe 4

Im Dateisystem werden unter `/dev` diverse Geräte aufgeführt, z. B. auch `/dev/mem` für den Hauptspeicher. Was gilt für solche Geräte?

( ) Sie haben keinen Besitzer.

( ) Es sind keine Lese-/Schreib-/Ausführungsrechte setzbar.

( ) Alle in `/dev` aufgeführten Geräte müssen vor der ersten Nutzung gemountet werden.

(X) Sie werden über Gerätetreiber angesprochen, sofern es sich nicht um so einfache Pseudogeräte wie `/dev/null` handelt.

( ) Die in `/dev` aufgeführten Namen, z. B. `mem`, `hd`, `lp`, ..., sind geschützte Bezeichner und dürfen nicht für eigene Dateien verwendet werden.

Aufgabe 5

In unserer Sammlung gefährlicher und dummer Dateinamen betrachten wir heute den Namen, der nur aus dem sonst für die Umlenkung genutzten Zeichen „>“ besteht. Kann man ihn erzeugen?

(X) Ja, kann man z. B. mit `touch \>` erzeugen.

(X) Ja, kann man z. B. mit `touch ">"` erzeugen.

( ) Ja, kann man sogar ganz locker mit `touch >` erzeugen.

(X) Ja, kann man mit `touch '>'` erzeugen.

( ) Nein, das Größerzeichen („>“) ist als Dateiname nicht erzeugbar und verboten, weil es für die Umlenkung in der Shell gebraucht wird.

Aufgabe 6

Kann ein User einen Prozess mit der Prozess-ID 1 erzeugen?

( ) Ja, ist möglich und kann von dem User erzwungen werden.

( ) Ja, ist möglich, wenn dieser noch nicht benutzt wird.

(X) Nein, ist nicht möglich. (`init` hat immer die PID 1)

( ) Ja, aber nur als Super-User.

( ) Es gibt unter Unix/Linux keine Prozess-IDs mehr.

Aufgabe 7

Benutzer `fix` möchte die `datei1` in `datei2` mittels `mv datei1 datei2` umbenennen. Welche Kommandofolgen haben den gleichen Effekt?

(X) `cp datei1 datei2; rm datei1`

(X) `cp datei1 datei2 && rm datei1`

( ) `echo -n >datei1; echo -n >datei2`

(X) `ln datei1 datei2; rm datei1`

( ) `mv datei1 .`

Aufgabe 8

Fügen Sie der Variablen `PATH` das `HOME`-Verzeichnis hinzu!

(X) `PATH=$PATH:$HOME`

( ) `$PATH=$PATH:$HOME`

( ) `PATH+=$HOME`

( ) `PATH=$PATH;$HOME`

( ) Geht grundsätzlich nicht, das `HOME`-Verzeichnis lässt sich nicht zu `PATH` hinzufügen.

Aufgabe 9

Eine Datei löscht man mit dem `rm`-Kommando.

(X) Ja.

( ) Nur wenn die Datei nicht versteckt ist.

( ) Nur wenn die Datei schon leer ist, sonst `rm -r`.

( ) Kommt auf die Größe der Datei an.

( ) Nein, geht mit dem `rmdir`-Kommando.

Aufgabe 10

Was wird mit dem `set`-Kommando ohne Argumente angezeigt?

( ) Nur die für den Export freigegebenen Variablen.

( ) Nur die für den Import freigegebenen Variablen.

(X) Alle in der Shell bekannten Variablen.

( ) Nur die read-only Variablen.

( ) Nur die Optionen, mit denen die gegenwärtige Shell aufgerufen wurde.

Aufgabe 11

Welche Aussagen sind für die Kommandoverknüpfung *Kommando1* `||` *Kommando2* richtig?

( ) Beide Kommandos werden in jedem Fall ausgeführt.

(X) *Kommando2* wird nur ausgeführt, wenn *Kommando1* fehlschlägt.

( ) *Kommando2* wird nur ausgeführt, wenn *Kommando1* erfolgreich ausgeführt wird.

( ) *Kommando2* wird nur ausgeführt, wenn *Kommando1* eine nicht-leere Zeichenfolge in die Pipe schickt, unabhängig vom Exit-Status, den *Kommando1* zurückliefert.

( ) Ersetzt die Kommandoverknüpfung *Kommando1*`;` *Kommando2* mit gleicher Wirkung.

Aufgabe 12

Welche Kommandozeilen erzeugen dieselbe Ausgabe wie der Aufruf von `date`?

( ) `echo date`

(X) `echo ‘date‘`

(X) `date | cat`

( ) `cat ‘date‘`

( ) `cat date`

Aufgabe 13

Die Dateirechte am Shell-Skript `here` aus Aufgabe 14 seien 644. Wie lautet das `chmod`-Kommando, damit Besitzer `fix` es ausführen kann?

chmod +x here (und andere Möglichkeiten)

Aufgabe 14

Gegeben sei das folgende Shell-Skript `here`

```
sh <<ENDE
ls -l
ENDE
```

Welche Ausgabe erzeugt der Aufruf von `./here` ?

( ) Die Zeile `ls -l`.

(X) Eine Auflistung der Dateien im gegenwärtigen Verzeichnis wie bei `ls -l`.

( ) Eine Fehlermeldung `Datei ls nicht gefunden`.

( ) Eine Fehlermeldung `Datei ENDE nicht gefunden`.

( ) Die Dateieigenschaften für die Datei `ENDE` im Langformat.

Aufgabe 15

Gegeben sei das folgende Shell-Skript `zensur`.

```
for word do
    if [ $word = $1 ]
    then
        echo "*****"
    else
        echo $word
    fi
done
```

Was liefert der Aufruf mit `./zensur hallo aber hallo aber ja bitte` ?

****

aber

****

aber

ja

bitte

Aufgabe 16

Betrachten Sie die folgende Kommandoabfolge des Benutzers `fix`.

```
fix@labserv:~$ id
uid=1006(fix) gid=1009(fix) Gruppen=1009(fix),100(users)
fix@labserv:~$ echo $$
23029
fix@labserv:~$ newgrp users
fix@labserv:~$ id
uid=1006(fix) gid=100(users) Gruppen=1009(fix),100(users)
fix@labserv:~$ echo $$
26801
fix@labserv:~$ exit
fix@labserv:~$ id
uid=1006(fix) gid=1009(fix) Gruppen=1009(fix),100(users)
```

Was kann daraus geschlossen werden?

(X) Der Benutzer `fix` ändert temporär seine Gruppenkennung zu `users`.

(X) Es wird eine neue Untershell erzeugt.

( ) Der Gruppenwechsel erfolgt in der alten Shell.

(X) Um in die alte Gruppenidentität zurückzuwechseln, genügt ein CTRL-D (`exit`).

( ) Um in die alte Gruppenidentität zurückzuwechseln, muss `oldgrp fix` aufgerufen werden.

Aufgabe 17

Betrachten Sie die folgende Kommandofolge. Ergänzen Sie die fehlende Zeile!

```
fix@labserv:~$ ps
PID TTY          TIME CMD
23029 pts/6    00:00:00 bash
29213 pts/6    00:00:00 ps
fix@labserv:~$ bash
fix@labserv:~$ ps
PID TTY          TIME CMD
23029 pts/6    00:00:00 bash
29217 pts/6    00:00:00 bash
29220 pts/6    00:00:00 ps
fix@labserv:~$ echo $$ $#

___29217    0___
```

Aufgabe 18

Was erzeugt der Systemaufruf `fork()`, wenn er in einem Prozess *P* aufgerufen wird?

(X) Einen neuen, zu *P* identischen Prozess mit neuer Prozess-ID.

( ) Einen neuen Prozess *Q*, der *P* überlagert.

( ) Ein neues Textsegment mit gleicher Prozess-ID.

( ) Eine leere Prozess-ID ohne Text- und Datensegment (Pseudo-Zombie).

( ) Eine Änderung der User-Bits an *P* (SUID-Bit wird auf 1 gesetzt).

Aufgabe 19

Wir betrachten die Pipeline `ls -i | sort`. Was gilt bezüglich der Ausgabe?

( ) Das `ls`-Kommando wird im interaktiven Modus gestartet und verlangt die Benutzerfreigabe für jeden Eintrag.

(X) Dateinamen mit gleicher i-Nummer (aus `ln`) stehen hintereinander.

(X) Die Zeilen der Ausgabe sind aufsteigend nach i-Nummer angeordnet.

( ) Die Ausgabe erfolgt aufsteigend nach Dateinamen.

( ) Enthält auch die versteckten Dateien „`.`“ und „`..`“.

Aufgabe 20

Welche der folgenden Angaben zu einer Datei werden in den Verzeichnissen (nicht im i-node) selbst abgespeichert?

(X) Der Name der Datei.

(X) Die i-Nummer der Datei (Index des i-node).

( ) Der Besitzer.

( ) Die Dateirechte.

( ) Die Startadressen der Datenblöcke.

Aufgabe 21

Betrachten Sie das Shell-Script `append`. Bei Aufruf mit nur einem Dateinamen als Parameter soll die Eingabe aus `stdin` an die genannte *Zieldatei* angehängt werden. Bei Aufruf mit zwei Dateinamen soll der Inhalt von *Quelldatei* an *Zieldatei* angehangen werden. Füllen Sie die Lücken!

Lösungsvorschlag:

```
if [ _ $# _ -eq _ 1 _ ]
then cat __ >> __ $1
else  if [ __$# -eq 2 __ ]
    then cat  >> $2 < $1
    else echo "Anleitung: append [Quelldatei] Zieldatei"
    fi
fi
```

(und andere Möglichkeiten)

ENDE DER KLAUSUR

Prof. Dr. Lutz Wegner
Stefan Achler, M.Sc.

26. Juli 2013

Einführung in UNIX
Klausur zum Sommersemester 2013

Name:.....Vorname:.....

Matr. Nr.:.....Studiengang:.....

Bearbeiten Sie alle Fragen! Ankreuzaufgaben gibt es in dieser Klausur nicht mehr, da ihre Verwendung derzeit umstritten ist. Hilfsmittel sind nicht zugelassen! Falls Sie für die Beantwortung der Aufgaben zusätzlichen Platz benötigen, verwenden Sie die Rückseiten. Separate Blätter sind nicht zugelassen.

Aufgabe	Punkte max.	Punkte erreicht
1	2	
2	2	
3	2	
4	2	
5	2	
6	2	
7	2	
8	2	
9	2	
10	2	
11	2	
12	2	
13	2	
14	2	
15	2	
16	2	
17	2	
18	2	
19	2	
20	2	
Summe	**40**	

Aufgabe 1

Wie lautet der Kommandoname der ältesten Shell, auch als Bourne-Shell bekannt:

_____

Aufgabe 2

In welcher Datei im Heimatverzeichnis werden traditionell die Einstellungen abgelegt, die nach dem Login automatisch gesetzt werden sollen.

_____

Aufgabe 3

Mit welchem Kommando wird die in Aufgabe 2 gesuchte Datei ausgeführt, damit die Änderungen in der gegenwärtigen Shell gelten?

_____

Aufgabe 4

In unserer Sammlung gefährlicher und dummer Dateinamen betrachten wir heute den Namen, der nur aus einem einzelnen Hochkomma (single quote) „'“ besteht. Erzeugen Sie ihn mit dem folgenden `echo`-Kommando oder geben Sie an, dass er nicht erzeugt werden kann!

```
echo dummes Zeug > _____
```

Aufgabe 5

Zwei Anwender starten das selbe Kommando. Welches Segment können Sie sich teilen?

_____

Aufgabe 6

Welcher Wert des Linkzählers muss erreicht sein, damit ein *Verzeichnis* gelöscht wird?

Linkzählerwert: _____

Aufgabe 7

Welcher Systemaufruf erzeugt zunächt einen Klon des aufrufenden Prozesses?

_____

Aufgabe 8

Mit welchem Systemaufruf erfolgt die Überlagerung des Prozessabbilds des in Aufgabe 7 erzeugten Klones?

_____

Aufgabe 9

Warum hat das Setzen des SUID-Bits an einem Shell-Skript keine Auswirkung?

_____

_____

Aufgabe 10

Sie rufen das ps-Kommando auf der Korn-Shell (ksh) heraus auf. Welche zwei Prozesse werden Sie mit Sicherheit sehen?

_____ und _____

Aufgabe 11

Nennen Sie ein (Spezial-)Kommando, das zwingend in jeder Shell ein sog. shell-internes Kommando (builtin command) ist!

_____

Aufgabe 12

Wie heißt die Spezialdatei (mit Pfadangabe), in der man unbenötigte Ausgaben verschwinden lassen kann und die sofort EOF liefert, wenn man aus ihr liest?

_____

Aufgabe 13

Gehört man mehreren Gruppen an, dann möchte man manchmal beim Arbeiten am Rechner seine Gruppenidentität wechseln. Mit welchem Kommando geht das?

_____

Aufgabe 14

Betrachten Sie den folgenden Ablauf bei Aufruf des Shell-Skripts `memo` und ergänzen Sie `memo`!

```
fix@labserv:~/testdir$ ./memo
Anrede eingeben: Herr Prof. Fix
Email-Adresse eingeben: fix
Kurznachricht eingeben:
Klausur ist leicht!
fix@labserv:~/testdir$ mail
"/var/mail/fix": 1 message 1 new
>N  1 fix@labserv.db.in  Wed Jul 17 13:40   14/603
& p
To: fix@labserv.db.informatik.uni-kassel.de
Date: Wed, 17 Jul 2013 13:40:40 +0200 (CEST)
From: fix@labserv.db.informatik.uni-kassel.de
Hallo Herr Prof. Fix
Klausur ist leicht!
& q
...
```

memo

```
echo -n "Anrede eingeben: "
read anrede
echo -n "Email-Adresse eingeben: "
read email
echo "Kurznachricht eingeben: "
read nachricht
echo Hallo _____________________________

   _____________________________________
```

Aufgabe 15

Ergänzen Sie das folgende `echo`-Kommando, damit in der Ausgabe immer die gegenwärtige Datums- und Uhrzeitangabe steht.

```
echo Heute ist _____________________________
```

Aufgabe 16

Betrachten Sie den folgenden Ablauf. Was erzeugt das letzte Kommando?

```
fix@labserv:~/testdir$ Echo hallo 2>&1
-bash: Echo: Kommando nicht gefunden.
fix@labserv:~/testdir$ Echo hallo 2>1
```

_____

_____

Aufgabe 17

Was ist die Standardvorbelegung der Dateirechte für neu angelegte *Normaldateien* als Oktalzahl?

_____

Aufgabe 18

Wo stehen die in Aufgabe 17 genannten Dateirechte?

_____

Aufgabe 19

Welche Ausgabe erzeugt das folgende Shell-Skript `meinekatze`?

meinekatze

```
tr -s ' ' '_' <<HERE
pi pa po
HERE
```

_____

Aufgabe 20

Aufgerufen werde ein Shell-Skript `./klausur jetzt ist aber schluss!`

Wie lautet die Belegung der folgenden Parameter?

`$0` _____ `$#` _____

`$*` _____

ENDE DER KLAUSUR

Prof. Dr. Lutz Wegner
Stefan Achler, M.Sc.

26. Juli 2013

Einführung in UNIX
Klausur zum Sommersemester 2013

MUSTERLÖSUNG

Name:.....Vorname:.....

Matr. Nr.:.....Studiengang:.....

Bearbeiten Sie alle Fragen! Ankreuzaufgaben gibt es in dieser Klausur nicht mehr, da ihre Verwendung derzeit umstritten ist. Hilfsmittel sind nicht zugelassen! Falls Sie für die Beantwortung der Aufgaben zusätzlichen Platz benötigen, verwenden Sie die Rückseiten. Separate Blätter sind nicht zugelassen.

Aufgabe	Punkte max.	Punkte erreicht
1	2	
2	2	
3	2	
4	2	
5	2	
6	2	
7	2	
8	2	
9	2	
10	2	
11	2	
12	2	
13	2	
14	2	
15	2	
16	2	
17	2	
18	2	
19	2	
20	2	
Summe	**40**	

Aufgabe 1

Wie lautet der Kommandoname der ältesten Shell, auch als Bourne-Shell bekannt:

_____**sh**_____

Aufgabe 2

In welcher Datei im Heimatverzeichnis werden traditionell die Einstellungen abgelegt, die nach dem Login automatisch gesetzt werden sollen.

_____**.profile**_____

Aufgabe 3

Mit welchem Kommando wird die in Aufgabe 2 gesuchte Datei ausgeführt, damit die Änderungen in der gegenwärtigen Shell gelten?

___. ***oder*** **source**_____

Aufgabe 4

In unserer Sammlung gefährlicher und dummer Dateinamen betrachten wir heute den Namen, der nur aus einem einzelnen Hochkomma (single quote) „'“ besteht. Erzeugen Sie ihn mit dem folgenden `echo`-Kommando oder geben Sie an, dass er nicht erzeugt werden kann!

`echo dummes Zeug >` **__\'__** ***auch*** **__"'"__**

Aufgabe 5

Zwei Anwender starten das selbe Kommando. Welches Segment können Sie sich teilen?

____Textsegment___ ***auch*** **__Code, Programmcode__**

Aufgabe 6

Welcher Wert des Linkzählers muss erreicht sein, damit ein *Verzeichnis* gelöscht wird?

Linkzählerwert: _____**1**_____

Aufgabe 7

Welcher Systemaufruf erzeugt zunächt einen Klon des aufrufenden Prozesses?

_____**fork**_____

Aufgabe 8

Mit welchem Systemaufruf erfolgt die Überlagerung des Prozessabbilds des in Aufgabe 7 erzeugten Klones?

_____**exec**_____

Aufgabe 9

Warum hat das Setzen des SUID-Bits an einem Shell-Skript keine Auswirkung?

___Ändern des effektiven User-Ids wird als Nebeneffekt des _____

___exec-Systemaufrufs gemacht, den es bei Shell-Skripten nicht gibt.___

Aufgabe 10

Sie rufen das ps-Kommando auf der Korn-Shell (ksh) heraus auf. Welche zwei Prozesse werden Sie mit Sicherheit sehen?

_____**ps**_____ und _____**ksh**_____

Aufgabe 11

Nennen Sie ein (Spezial-)Kommando, das zwingend in jeder Shell ein sog. shell-internes Kommando (builtin command) ist!

___source, cd, break, read, ..._____

Aufgabe 12

Wie heißt die Spezialdatei (mit Pfadangabe), in der man unbenötigte Ausgaben verschwinden lassen kann und die sofort EOF liefert, wenn man aus ihr liest?

____/dev/null_____

Aufgabe 13

Gehört man mehreren Gruppen an, dann möchte man manchmal beim Arbeiten am Rechner seine Gruppenidentität wechseln. Mit welchem Kommando geht das?

____**newgrp**_____

Aufgabe 14

Betrachten Sie den folgenden Ablauf bei Aufruf des Shell-Skripts `memo` und ergänzen Sie `memo`!

```
fix@labserv:~/testdir$ ./memo
Anrede eingeben: Herr Prof. Fix
Email-Adresse eingeben: fix
Kurznachricht eingeben:
Klausur ist leicht!
fix@labserv:~/testdir$ mail
"/var/mail/fix": 1 message 1 new
>N  1 fix@labserv.db.in  Wed Jul 17 13:40   14/603
& p
To: fix@labserv.db.informatik.uni-kassel.de
Date: Wed, 17 Jul 2013 13:40:40 +0200 (CEST)
From: fix@labserv.db.informatik.uni-kassel.de
Hallo Herr Prof. Fix
Klausur ist leicht!
& q
...
```

memo

```
echo -n "Anrede eingeben: "
read anrede
echo -n "Email-Adresse eingeben: "
read email
echo "Kurznachricht eingeben: "
read nachricht
echo Hallo $anrede "
$nachricht" | mail $email
```

Aufgabe 15

Ergänzen Sie das folgende `echo`-Kommando, damit in der Ausgabe immer die gegenwärtige Datums- und Uhrzeitangabe steht.

`echo Heute ist ____`**`‘date‘`**`__`**(schräggestellte Anführungszeichen ‘!)**

Aufgabe 16

Betrachten Sie den folgenden Ablauf. Was erzeugt das letzte Kommando?

```
fix@labserv:~/testdir$ Echo hallo 2>&1
-bash: Echo: Kommando nicht gefunden.
fix@labserv:~/testdir$ Echo hallo 2>1
```

__eine Datei mit Namen 1 und dem ___

__Inhalt -bash: Echo: Kommando nicht gefunden._____

Aufgabe 17

Was ist die Standardvorbelegung der Dateirechte für neu angelegte *Normaldateien* als Oktalzahl?

_____**644**_____

Aufgabe 18

Wo stehen die in Aufgabe 17 genannten Dateirechte?

_____**im i-Knoten**_____

Aufgabe 19

Welche Ausgabe erzeugt das folgende Shell-Skript `meinekatze`?

meinekatze

```
tr -s ' ' '_' <<HERE
pi pa po
HERE
```

____**pi_pa_po**_____

Aufgabe 20

Aufgerufen werde ein Shell-Skript `./klausur jetzt ist aber schluss!`

Wie lautet die Belegung der folgenden Parameter?

`$0` _____**klausur**____ `$#` _____**4**_____

`$*` **__jetzt ist aber schluss!**_____

ENDE DER KLAUSUR

Prof. Dr. Lutz Wegner
Stefan Achler, M.Sc.

10. Februar 2014

Einführung in UNIX
Klausur zum Wintersemester 2013/14

Name:.....Vorname:.....

Matr. Nr.:.....Studiengang:.....

Bearbeiten Sie alle Fragen! Ankreuzaufgaben gibt es in dieser Klausur nicht, da ihre Verwendung immer noch umstritten ist. Hilfsmittel sind nicht zugelassen! Falls Sie für die Beantwortung der Aufgaben zusätzlichen Platz benötigen, verwenden Sie die Rückseiten. Separate Blätter sind nicht zugelassen. Die Bearbeitungszeit beträgt 60 Minuten.

Aufgabe	Punkte max.	Punkte erreicht
1	2	
2	2	
3	2	
4	2	
5	2	
6	2	
7	2	
8	2	
9	2	
10	2	
11	2	
12	2	
13	2	
14	2	
15	2	
16	2	
17	2	
18	2	
Summe	**36**	

Aufgabe 1

Auf der Wikipedia-Seite zur Unix-Shell findet sich das folgende Programm `mgruss`, wobei `+%H` eine Formatoption des `date`-Kommandos ist, die nur die Uhrzeit in Stunden liefert.

```
#!/bin/sh
tageszeit=`date +%H`
if [ $tageszeit -lt 12 ]; then
echo "Guten Morgen."
else echo "Guten Tag."
fi
```

Welche Shell wird hier in der Shell-Direktive `#!/bin/sh` aufgerufen?

Aufgabe 2

Wie lautet die `if`-Anweisung im Shell-Skript `mgruss` aus Aufgabe 1 oben, wenn man auf die Verwendung der Variablen `tageszeit` verzichten möchte?

Aufgabe 3

Welche Berechtigungen (Lese-, Schreib-, Ausführungsrecht) müssen für den Besitzer des Shell-Skripts `mgruss` gesetzt sein, damit das Shell-Skript mittels Aufruf `./mgruss` ausgeführt werden kann?

Aufgabe 4

In unserer Sammlung gefährlicher und dummer Dateinamen betrachten wir heute den Namen, der nur aus einem Neue-Zeile-Zeichen (erzeugt mit der Return-Taste) besteht. Wir erzeugen die Datei mit dem folgenden `echo`-Kommando und setzen das Ausführungsrecht. Führen Sie die Datei aus und geben Sie die Ausgabe an!

```
fix@labserv:~$ echo "echo hallo" >'
> '
fix@labserv:~$ chmod +x '
> '
fix@labserv:~$ _____________________________________

_____________________________

_____________________________
```

Aufgabe 5

Weisen zwei oder mehr Dateinamen per hartem Link auf dieselbe Datei, dann kann man für jeden (Alias-)Namen unterschiedliche Rechte setzen. Richtig oder falsch? Begründung!

Aufgabe 6

Welche Ausgaben liefert das Kommando `wc` in den Situationen unten?

```
fix@labserv:~$ echo -n >leer; wc leer
_____________ leer
fix@labserv:~$ echo -n " " >blank; wc blank
_____________ blank
fix@labserv:~$ echo -n hallo >hallo; wc hallo
_____________ hallo
fix@labserv:~$ echo hallo >line; wc line
_____________ line
```

Aufgabe 7

Selbstmodifizierender Code ist heute bei Programmen, die als Binärcode (executable) vorliegen, nicht üblich, weil die Textsegmente read-only sind und so von mehreren Anwendern gleichzeitig genutzt werden können. Ein selbstmodifizierendes Shell-Skript ist aber kein Problem. Ergänzen Sie die Lücke so, dass der Effekt auch für alle Aufrufsituationen (Aliasnamen, anderes Verzeichnis) funktioniert!

```
fix@labserv:~$ more selbstmodifizierend
echo "hallo Leute"
echo "echo 'noch eine Zeile'" _____________________________
fix@labserv:~$ chmod +x selbstmodifizierend
fix@labserv:~$ ./selbstmodifizierend
hallo Leute
noch eine Zeile
fix@labserv:~$ ./selbstmodifizierend
hallo Leute
noch eine Zeile
noch eine Zeile
fix@labserv:~$
```

Aufgabe 8

Ein Prozess kann mit mehreren `fork`-Systemaufrufen eine Reihe von Kinderprozessen erzeugen, die sich ggf. mit unterschiedlichem Programmcode überlagern. Wie kann der Vaterprozess zwischen seinen Kinderprozessen unterscheiden, z.B. bei einem `wait`?

Aufgabe 9

Wie sieht die Anzeige der Rechte bei `passwd` in der Situation unten aus, d.h. was steht an Stelle der drei Fragezeichen?

```
fix@labserv:~$ which passwd

/usr/bin/passwd

fix@labserv:~$ ls -l /usr/bin/passwd

-???r-xr-x 1 root root 43280 15. Feb 2011  /usr/bin/passwd
```

Aufgabe 10

Kann ein Anwender auch in einer Mehrprozessumgebung sicher sein, dass er mit aufeinanderfolgenden Kommandos linear aufsteigende Prozessidentifier (PID) ohne Lücken erhält? Kurze Begründung!

Aufgabe 11

Wie heißt der Prozess, der alle Waisenkinder adoptiert und welche Prozess-ID hat er?

Aufgabe 12

Ergänzen Sie den Aufruf des Shell-Skripts `abmelden`, damit die unten gezeigte Ausgabe erscheint (es waren nur *achler* und *fix* angemeldet).

```
fix@labserv:~$ more abmelden
read a b
while [ "$a" != "" ]
do
  echo "$a" Zeit heimzugehen!
  read a b
done
fix@labserv:~$ _____________________ ./abmelden
achler Zeit heimzugehen!
fix Zeit heimzugehen!
fix@labserv:~$
```

Aufgabe 13

Mal angenommen, die angemeldeten Teilnehmer hätten Botschaften auf ihr Terminal zugelassen (*write permission turned on*), was wäre im Skript `abmelden` zu ändern gewesen, damit den Teilnehmern die Botschaft aufs Terminal gesendet würde? Ergänzen Sie die betreffende Zeile um den zusätzlichen Befehl!

Aufgabe 14

Mit dem Kommando `tty` kann man sich den Namen des gegenwärtigen Terminals anzeigen lassen. Ergänzen Sie das fehlende Kommando unten! Hinweis: Es ist nicht `chmod`!

```
fix@labserv:~$ tty

/dev/pts/1

fix@labserv:~$ ls -l /dev/pts/1

crw------- 1 fix tty 136, 1 28. Jan 11:45 /dev/pts/1


fix@labserv:~$ _________________ y

fix@labserv:~$ ls -l /dev/pts/1

crw--w---- 1 fix tty 136, 1 28. Jan 11:45 /dev/pts/1

fix@labserv:~$
```

Aufgabe 15

Wie lautet der volle Pfadname für das Terminal, das als generisches Gerät der gegenwärtigen Shell zugeordnet ist?

Aufgabe 16

Mit `ls -u` *Verzeichnis* erhält man die Auflistung der Dateien in *Verzeichnis* sortiert nach Zugriffszeit, neueste Zeit zuerst. Es werde `ls -u >ganzfrisch` aufgerufen. Ist der Dateiname `ganzfrisch` jetzt der erste Eintrag in Datei `ganzfrisch`? Ja oder nein – jeweils mit Begründung!

Aufgabe 17

Prof. Fix kennt das Passwort von `root` und möchte temporär aus seiner Shell heraus die Identität von `root` annehmen (den User-ID auf root setzen). Mit welchem Kommando gelingt dies?

Aufgabe 18

Aus Sicherheitsgründen stellt man das gegenwärtige Verzeichnis nicht mehr an den Anfang des Kommandosuchpfads. Mit welcher Eingabe würde man den Suchpfad trotzdem so setzen?

ENDE DER KLAUSUR

Prof. Dr. Lutz Wegner
Stefan Achler, M.Sc.

10. Februar 2014

Einführung in UNIX
Klausur zum Wintersemester 2013/14

MUSTERLÖSUNG

Name:.....Vorname:.....

Matr. Nr.:.....Studiengang:.....

Bearbeiten Sie alle Fragen! Ankreuzaufgaben gibt es in dieser Klausur nicht, da ihre Verwendung immer noch umstritten ist. Hilfsmittel sind nicht zugelassen! Falls Sie für die Beantwortung der Aufgaben zusätzlichen Platz benötigen, verwenden Sie die Rückseiten. Separate Blätter sind nicht zugelassen. Die Bearbeitungszeit beträgt 60 Minuten.

Aufgabe	Punkte max.	Punkte erreicht
1	2	
2	2	
3	2	
4	2	
5	2	
6	2	
7	2	
8	2	
9	2	
10	2	
11	2	
12	2	
13	2	
14	2	
15	2	
16	2	
17	2	
18	2	
Summe	**36**	

Aufgabe 1

Auf der Wikipedia-Seite zur Unix-Shell findet sich das folgende Programm `mgruss`, wobei `+%H` eine Formatoption des `date`-Kommandos ist, die nur die Uhrzeit in Stunden liefert.

```
#!/bin/sh
tageszeit=`date +%H`
if [ $tageszeit -lt 12 ]; then
echo "Guten Morgen."
else echo "Guten Tag."
fi
```

Welche Shell wird hier in der Shell-Direktive `#!/bin/sh` aufgerufen?

Die Bourne-Shell

Aufgabe 2

Wie lautet die `if`-Anweisung im Shell-Skript `mgruss` aus Aufgabe 1 oben, wenn man auf die Verwendung der Variablen `tageszeit` verzichten möchte?

if [`date +%H` -lt 12]; then

Aufgabe 3

Welche Berechtigungen (Lese-, Schreib-, Ausführungsrecht) müssen für den Besitzer des Shell-Skripts `mgruss` gesetzt sein, damit das Shell-Skript mittels Aufruf `./mgruss` ausgeführt werden kann?

Lese- und Ausführungsrecht

Aufgabe 4

In unserer Sammlung gefährlicher und dummer Dateinamen betrachten wir heute den Namen, der nur aus einem Neue-Zeile-Zeichen (erzeugt mit der Return-Taste) besteht. Wir erzeugen die Datei mit dem folgenden `echo`-Kommando und setzen das Ausführungsrecht. Führen Sie die Datei aus und geben Sie die Ausgabe an!

```
fix@labserv:~$ echo "echo hallo" >'
> '
fix@labserv:~$ chmod +x '
> '
fix@labserv:~$ ./'_____________________________

> '_____________________

hallo_____________________
```

Aufgabe 5

Weisen zwei oder mehr Dateinamen per hartem Link auf dieselbe Datei, dann kann man für jeden (Alias-)Namen unterschiedliche Rechte setzen. Richtig oder falsch? Begründung!

Falsch, die Rechte werden im i-Knoten abgespeichert, den es je Datei genau einmal gibt.

Aufgabe 6

Welche Ausgaben liefert das Kommando `wc` in den Situationen unten?

```
fix@labserv:~$ echo -n >leer; wc leer
0 0 0_______ leer
fix@labserv:~$ echo -n " " >blank; wc blank
0 0 1_______ blank
fix@labserv:~$ echo -n hallo >hallo; wc hallo
0 1 5_______ hallo
fix@labserv:~$ echo hallo >line; wc line
1 1 6_______ line
```

Aufgabe 7

Selbstmodifizierender Code ist heute bei Programmen, die als Binärcode (executable) vorliegen, nicht üblich, weil die Textsegmente read-only sind und so von mehreren Anwendern gleichzeitig genutzt werden können. Ein selbstmodifizierendes Shell-Skript ist aber kein Problem. Ergänzen Sie die Lücke so, dass der Effekt auch für alle Aufrufsituationen (Aliasnamen, anderes Verzeichnis) funktioniert!

```
fix@labserv:~$ more selbstmodifizierend
echo "hallo Leute"
echo "echo 'noch eine Zeile'" >> $0___________________
fix@labserv:~$ chmod +x selbstmodifizierend
fix@labserv:~$ ./selbstmodifizierend
hallo Leute
noch eine Zeile
fix@labserv:~$ ./selbstmodifizierend
hallo Leute
noch eine Zeile
noch eine Zeile
fix@labserv:~$
```

Aufgabe 8

Ein Prozess kann mit mehreren `fork`-Systemaufrufen eine Reihe von Kinderprozessen erzeugen, die sich ggf. mit unterschiedlichem Programmcode überlagern. Wie kann der Vaterprozess zwischen seinen Kinderprozessen unterscheiden, z.B. bei einem `wait`?

Der Vaterprozess muss sich den bei fork zurückgelieferten Prozessidentifier merken. Dieser kennzeichnet den erzeugten Kindprozess. Mit diesem PID kann als Argument von wait auf die Beendigung genau dieses Prozesses gewartet werden.

Aufgabe 9

Wie sieht die Anzeige der Rechte bei `passwd` in der Situation unten aus, d.h. was steht an Stelle der drei Fragezeichen?

```
fix@labserv:~$ which passwd

/usr/bin/passwd

fix@labserv:~$ ls -l /usr/bin/passwd

-???r-xr-x 1 root root 43280 15. Feb 2011  /usr/bin/passwd
```

_____rws (SUID-Bit gesetzt)_____

Aufgabe 10

Kann ein Anwender auch in einer Mehrprozessumgebung sicher sein, dass er mit aufeinanderfolgenden Kommandos linear aufsteigende Prozessidentifier (PID) ohne Lücken erhält? Kurze Begründung!

Nein, es werden auch alte PIDs wiederverwertet und andere Prozesse beanspruchen (verbrauchen) auch PIDs.

Aufgabe 11

Wie heißt der Prozess, der alle Waisenkinder adoptiert und welche Prozess-ID hat er?

init mit PID 1

Aufgabe 12

Ergänzen Sie den Aufruf des Shell-Skripts `abmelden`, damit die unten gezeigte Ausgabe erscheint (es waren nur *achler* und *fix* angemeldet).

```
fix@labserv:~$ more abmelden
read a b
while [ "$a" != "" ]
do
  echo "$a" Zeit heimzugehen!
  read a b
done
fix@labserv:~$ _____who | ________ ./abmelden
achler Zeit heimzugehen!
fix Zeit heimzugehen!
fix@labserv:~$
```

Aufgabe 13

Mal angenommen, die angemeldeten Teilnehmer hätten Botschaften auf ihr Terminal zugelassen (*write permission turned on*), was wäre im Skript `abmelden` zu ändern gewesen, damit den Teilnehmern die Botschaft aufs Terminal gesendet würde? Ergänzen Sie die betreffende Zeile um den zusätzlichen Befehl!

`echo "$a" Zeit heimzugehen! | write "$a"`

Aufgabe 14

Mit dem Kommando `tty` kann man sich den Namen des gegenwärtigen Terminals anzeigen lassen. Ergänzen Sie das fehlende Kommando unten! Hinweis: Es ist nicht `chmod`!

```
fix@labserv:~$ tty

/dev/pts/1

fix@labserv:~$ ls -l /dev/pts/1

crw------- 1 fix tty 136, 1 28. Jan 11:45 /dev/pts/1


fix@labserv:~$ _____mesg____ y

fix@labserv:~$ ls -l /dev/pts/1

crw--w---- 1 fix tty 136, 1 28. Jan 11:45 /dev/pts/1

fix@labserv:~$
```

Aufgabe 15

Wie lautet der volle Pfadname für das Terminal, das als generisches Gerät der gegenwärtigen Shell zugeordnet ist?

/dev/tty

Aufgabe 16

Mit `ls -u` *Verzeichnis* erhält man die Auflistung der Dateien in *Verzeichnis* sortiert nach Zugriffszeit, neueste Zeit zuerst. Es werde `ls -u >ganzfrisch` aufgerufen. Ist der Dateiname `ganzfrisch` jetzt der erste Eintrag in Datei `ganzfrisch`? Ja oder nein – jeweils mit Begründung!

Ja, die Shell organisiert die Umlenkung, d.h. ganzfrisch wird angelegt oder leergemacht, bevor das ls-Kommando zur Ausführung kommt.

Aufgabe 17

Prof. Fix kennt das Passwort von `root` und möchte temporär aus seiner Shell heraus die Identität von `root` annehmen (den User-ID auf root setzen). Mit welchem Kommando gelingt dies?

su

Aufgabe 18

Aus Sicherheitsgründen stellt man das gegenwärtige Verzeichnis nicht mehr an den Anfang des Kommandosuchpfads. Mit welcher Eingabe würde man den Suchpfad trotzdem so setzen?

PATH=.:$PATH oder auch nur PATH=:$PATH

ENDE DER KLAUSUR

Prof. Dr. Lutz Wegner
Stefan Achler, M.Sc.

1. August 2014

Einführung in UNIX
Klausur zum Sommersemester 2014

Name:.....Vorname:.....

Matr. Nr.:.....Studiengang:.....

Bearbeiten Sie alle Fragen! Ankreuzaufgaben gibt es in dieser Klausur nicht, da ihre Verwendung immer noch umstritten ist. Hilfsmittel sind nicht zugelassen! Falls Sie für die Beantwortung der Aufgaben zusätzlichen Platz benötigen, verwenden Sie die Rückseiten. Separate Blätter sind nicht zugelassen. Die Bearbeitungszeit beträgt 60 Minuten.

Aufgabe	Punkte max.	Punkte erreicht
1	2	
2	2	
3	2	
4	2	
5	2	
6	2 + 2	
7	2	
8	2	
9	2	
10	2	
11	2	
12	2	
13	2	
14	2	
15	2	
16	2	
17	2	
18	2	
19	2	
20	2	
Summe	**42**	

Aufgabe 1

Die NSA hat das normale `cat`-Kommando durch das folgende Skript ersetzt. Es verwendet das `tee`-Kommando mit der Option `-a` (append).

```
cat $* | tee -a nsa-log | cat
```

Wie sähe das Skript mit gleicher Funktion ohne `tee` aus? Füllen Sie die Lücke!

```
cat $* _____________________ cat $*
```

Aufgabe 2

Studierende, die sich die Klausur vom letzten Semester angeschaut haben, wissen, dass `date +%H` die Uhrzeit im Stundenformat liefert. Das `at`-Kommando liefert die Ausgabe des ausgeführten Skripts in der Mailbox ab. Was wird durch die folgende Kommandozeile in unserer Mailbox abgelegt?

```
echo "date +%H" | at 13:37
```

Aufgabe 3

Ruft man das `at`-Kommando ähnlich wie in Aufgabe 2 aus der `bash` heraus auf zur Überprüfung der beteiligten Prozesse, so erhält man zunächst die folgende Mitteilung

```
warning: commands will be executed using /bin/sh
```

und dann zur Ausführungszeit die folgende Ausgabe. Wie man sieht, wird auch noch ein `ssh`-Dämon angezeigt. Ergänzen Sie die Lücken!

```
 PID TTY          TIME CMD
6543 ?        00:00:00 sshd
6726 ?        00:00:00 _____________
6727 ?        00:00:00 _____________
```

Aufgabe 4

In unserer Sammlung gefährlicher und dummer Dateinamen betrachten wir heute den Namen, der aus einem Punkt und umschließenden einfachen Anführungszeichen besteht (damit 3 Zeichen insgesamt). Füllen Sie die Lücke zum Erzeugen dieser Datei!

```
fix@labserv:~$ echo >____________________
```

Aufgabe 5

Die in Aufgabe 4 erzeugte Datei wollen wir schnell wieder los werden. In der Eile schreiben wir

```
fix@labserv:~$ rm '.'
```

Wie lautet die Fehlermeldung sinngemäß?

Aufgabe 6

Man betrachte das folgende Shell-Skript `yes`.

```
# the yes-command
word=yes

if [ $# -ne 0  ]
    then word=$1
fi

while  true
do
    echo $word
    sleep 1s
done
```

(a) Was bewirkt der Aufruf mit `./yes ja` ?

(b) Was bewirkt der Aufruf mit `./yes | head -3` ?

Aufgabe 7

Erläutern Sie den Unterschied der beiden Aufrufe aus dem Verzeichnis `/home/fix` heraus.

```
fix@labserv:~$ ln -s . Punkt

fix@labserv:~$ ln -s `pwd` Punkt
```

Aufgabe 8

Was liefert der folgende Aufruf?

```
fix@labserv:~$ ln . Punkt
```

Aufgabe 9

Was gilt für den Prozess, der eine Datei (als executable) ausführt, die das Setuid-Bit gesetzt hat?

Aufgabe 10

Für eine Textdatei seien die Dateirechte auf oktal `644` gesetzt. Was bedeutet das für den Besitzer, die Gruppe und den Rest?

Aufgabe 11

Ein Prozess hat sich beendet, sein Vaterprozess hat allerdings noch kein *wait* gemacht. Kann das passieren? Wenn nein, begründen Sie wie Unix das verhindert! Wenn ja, wie nennt man so einen Prozess?

Aufgabe 12

Längere Ausgaben mit dem `cat`-Kommando rauschen für den menschlichen Leser oft zu schnell durch. Mit welchen beiden Kommandos unter Linux bekommt man mehr oder weniger elegant eine seitenweise Ausgabe?

Aufgabe 13

Wie heißt das Programm, das die Terminals verwaltet und nach dem Usernamen fragt?

Aufgabe 14

Der `cron`-Prozess wird beim Hochfahren des Rechners gestartet und schaut nach Kommandos, die regelmäßig zur Ausführung kommen müssen. Was ist der Oberbegriff in Unix für diese Art von Dienstprozessen.

Aufgabe 15

Was ist der Unterschied in der Wirkung von `kill 4711` und `kill -9 4711`.
Hinweis: Das Default-Signal von `kill` ist 15 (`SIGTERM`).

Aufgabe 16

Wie wird die Verknüpfung der Standardein-/ausgabe (Vorbelegung) für einen neu erzeugten Unterprozess geregelt?

Aufgabe 17

Warum ist das `read`-Kommando ein Spezialkommando der Shell (built-in command)? Kurze Erläuterung am Beispiel von `read eingabe`.

Aufgabe 18

Aus Sicherheitsgründen nimmt man das gegenwärtige Verzeichnis nicht mehr in den Kommandosuchpfad auf. Mit welcher Kommandozeile würde man es trotzdem an das Ende des Suchpfads setzen?

Aufgabe 19

Mit `touch myfile` kann man das Modifikationsdatum der Datei `myfile` auf das gegenwärtige Datum (und die gegenwärtige Uhrzeit) setzen. Wie könnte man das Datum aller Einträge (auch der versteckten) im gegenwärtigen Verzeichnis mit `touch` setzen?

Aufgabe 20

Was macht das folgende Shell-Skript `myremove`? Hinweis: `.Papierkorb` ist ein Verzeichnis in unserem Heimatverzeichnis.

```
if [ "$1" = "-f" ]
then shift 1
    rm -r $*
else mv $* $HOME/.Papierkorb
fi
```

ENDE DER KLAUSUR

Prof. Dr. Lutz Wegner
Stefan Achler, M.Sc.

1. August 2014

Einführung in UNIX
Klausur zum Sommersemester 2014

MUSTERLÖSUNG

Name:.....Vorname:.....

Matr. Nr.:.....Studiengang:.....

Bearbeiten Sie alle Fragen! Ankreuzaufgaben gibt es in dieser Klausur nicht, da ihre Verwendung immer noch umstritten ist. Hilfsmittel sind nicht zugelassen! Falls Sie für die Beantwortung der Aufgaben zusätzlichen Platz benötigen, verwenden Sie die Rückseiten. Separate Blätter sind nicht zugelassen. Die Bearbeitungszeit beträgt 60 Minuten.

Aufgabe	Punkte max.	Punkte erreicht
1	2	
2	2	
3	2	
4	2	
5	2	
6	2 + 2	
7	2	
8	2	
9	2	
10	2	
11	2	
12	2	
13	2	
14	2	
15	2	
16	2	
17	2	
18	2	
19	2	
20	2	
Summe	**42**	

Aufgabe 1

Die NSA hat das normale `cat`-Kommando durch das folgende Skript ersetzt. Es verwendet das `tee`-Kommando mit der Option `-a` (append).

```
cat $* | tee -a nsa-log | cat
```

Wie sähe das Skript mit gleicher Funktion ohne `tee` aus? Füllen Sie die Lücke!

```
cat $* __>> nsa-log ;__ cat $*
```

Aufgabe 2

Studierende, die sich die Klausur vom letzten Semester angeschaut haben, wissen, dass `date +%H` die Uhrzeit im Stundenformat liefert. Das `at`-Kommando sendet die Ausgabe des ausgeführten Skripts an die Mailbox. Was wird durch die folgende Kommandozeile in unserer Mailbox abgelegt?

```
echo "date +%H" | at 13:37
```

Eine Mail mit der Zahl 13.

Aufgabe 3

Ruft man das `at`-Kommando ähnlich wie in Aufgabe 2 aus der `bash` heraus auf zur Überprüfung der beteiligten Prozesse, so erhält man zunächst die folgende Mitteilung

```
warning: commands will be executed using /bin/sh
```

und dann zur Ausführungszeit die folgende Ausgabe. Wie man sieht, wird auch noch ein `ssh`-Dämon angezeigt. Ergänzen Sie die Lücken!

```
  PID TTY          TIME CMD
 6543 ?        00:00:00 sshd
 6726 ?        00:00:00 __sh__
 6727 ?        00:00:00 __ps__
```

Aufgabe 4

In unserer Sammlung gefährlicher und dummer Dateinamen betrachten wir heute den Namen, der aus einem Punkt und umschließenden einfachen Anführungszeichen besteht (damit 3 Zeichen insgesamt). Füllen Sie die Lücke zum Erzeugen dieser Datei!

```
fix@labserv:~$ echo >__'.'___ (auch \'.\')
```

Aufgabe 5

Die in Aufgabe 4 erzeugte Datei wollen wir schnell wieder los werden. In der Eile schreiben wir

```
fix@labserv:~$ rm '.'
```

Wie lautet die Fehlermeldung sinngemäß?

Kein Löschen eines Verzeichnisses mit rm.

Aufgabe 6

Man betrachte das folgende Shell-Skript `yes`.

```
# the yes-command
word=yes

if [ $# -ne 0  ]
    then word=$1
fi

while  true
do
    echo $word
    sleep 1s
done
```

(a) Was bewirkt der Aufruf mit `./yes ja` ?

Eine Folge von Zeilen mit dem Wort ja, jede im Abstand von einer Sekunde.

(b) Was bewirkt der Aufruf mit `./yes | head -3` ?

Drei Zeilen mit yes.

Aufgabe 7

Erläutern Sie den Unterschied der beiden Aufrufe aus dem Verzeichnis `/home/fix` heraus.

```
fix@labserv:~$ ln -s . Punkt

fix@labserv:~$ ln -s `pwd` Punkt
```

Ein symbolischer Link mit Namen *Punkt*, der auf das Verzeichnis . zeigt. Im zweiten Fall zeigt der symbolische Link auf das Verzeichnis */home/fix* (absoluter Pfadname).

Aufgabe 8

Was liefert der folgende Aufruf?

```
fix@labserv:~$ ln . Punkt
```

Eine Fehlermeldung „kein harter Link auf Verzeichnisse!"

Aufgabe 9

Was gilt für den Prozess, der eine Datei (als executable) ausführt, die das Setuid-Bit gesetzt hat?

Der Prozess hat als effektive User-Id die User-Id der Datei.

Aufgabe 10

Für eine Textdatei seien die Dateirechte auf oktal `644` gesetzt. Was bedeutet das für den Besitzer, die Gruppe und den Rest?

**Wie rw-r--r--, also Lesen und Schreiben für den Besitzer,
nur Lesen für Gruppe und Rest.**

Aufgabe 11

Ein Prozess hat sich beendet, sein Vaterprozess hat allerdings noch kein *wait* gemacht. Kann das passieren? Wenn nein, begründen Sie wie Unix das verhindert! Wenn ja, wie nennt man so einen Prozess?

Zombie-Prozess

Aufgabe 12

Längere Ausgaben mit dem `cat`-Kommando rauschen für den menschlichen Leser oft zu schnell durch. Mit welchen beiden Kommandos unter Linux bekommt man mehr oder weniger elegant eine seitenweise Ausgabe?

more* und *less

Aufgabe 13

Wie heißt das Programm, das die Terminals verwaltet und nach dem Usernamen fragt?

getty

Aufgabe 14

Der `cron`-Prozess wird beim Hochfahren des Rechners gestartet und schaut nach Kommandos, die regelmäßig zur Ausführung kommen müssen. Was ist der Oberbegriff in Unix für diese Art von Dienstprozessen.

Dämon-Prozess (engl. daemon process)

Aufgabe 15

Was ist der Unterschied in der Wirkung von `kill 4711` und `kill -9 4711`.
Hinweis: Das Default-Signal von `kill` ist 15 (`SIGTERM`).

Ohne Option -9 kann das Signal vom Prozess abgefangen werden, mit Option -9 wird SIGKILL gesendet, was nicht abfangbar ist.

Aufgabe 16

Wie wird die Verknüpfung der Standardein-/ausgabe (Vorbelegung) für einen neu erzeugten Unterprozess geregelt?

Vorbelegung wird vom Vaterprozess (in der Regel die Shell) vererbt.

Aufgabe 17

Warum ist das `read`-Kommando ein Spezialkommando der Shell (built-in command)? Kurze Erläuterung am Beispiel von `read eingabe`.

Die in *eingabe* eingelesenen Werte wären in der Shell, aus der das *read* aufgerufen wurde, nicht bekannt (Einbahnstraße der Werteübergabe).

Aufgabe 18

Aus Sicherheitsgründen nimmt man das gegenwärtige Verzeichnis nicht mehr in den Kommandosuchpfad auf. Mit welcher Kommandozeile würde man es trotzdem an das Ende des Suchpfads setzen?

PATH=$PATH: (auch PATH=$PATH:.)

Aufgabe 19

Mit `touch myfile` kann man das Modifikationsdatum der Datei `myfile` auf das gegenwärtige Datum (und die gegenwärtige Uhrzeit) setzen. Wie könnte man das Datum aller Einträge (auch der versteckten) im gegenwärtigen Verzeichnis mit `touch` setzen?

touch * .*

Aufgabe 20

Was macht das folgende Shell-Skript `myremove`? Hinweis: `.Papierkorb` ist ein Verzeichnis in unserem Heimatverzeichnis.

```
if [ "$1" = "-f" ]
then shift 1
    rm -r $*
else mv $* $HOME/.Papierkorb
fi
```

Wird *myremove* mit der Option „-f" aufgerufen, löscht es die angegebenen Dateien und Verzeichnisse rekursiv, sonst verschiebt es sie in das Verzeichnis .Papierkorb.

ENDE DER KLAUSUR

Prof. Dr. Lutz Wegner
Stefan Achler M.Sc. **26. Februar 2015**

Einführung in UNIX
Klausur zum Wintersemester 2014/15

Name:.....Vorname:.....

Matr. Nr.:.....Studiengang:.....

Bearbeiten Sie alle Fragen! Hilfsmittel sind nicht zugelassen! Falls Sie für die Beantwortung der Aufgaben zusätzlichen Platz benötigen, verwenden Sie die Rückseiten.

Aufgabe	Punkte max.	Punkte erreicht
1	2	
2	2	
3	2	
4	2	
5	2	
6	2	
7	2 + 2	
8	2	
9	2	
10	2	
11	2	
12	2	
13	2	
14	4	
15	2	
Summe	**34**	

Aufgabe 1

Das Pseudogerät `/dev/null` wird gerne als „Biteimer“ bezeichnet, weil dorthin umgeleitete Daten geräuschlos verschwinden. Kann man aus `/dev/null` auch lesen und wenn ja, was erhält man dann?

Aufgabe 2

Wo wir gerade bei `/dev/null` sind: Mit `ln /dev/null papierkorb` wollte sich Prof. Fix einen Link in seinem Heimatverzeichnis anlegen, was aber bei unserem System mit einer Fehlermeldung beantwortet wurde. Was besagt die Fehlermeldung sinngemäß?

Aufgabe 3

Für die längst überfällige Sprachanpassung ans Schwäbische haben wir das folgende Shellskript `heitisch` („heute-ist“) entworfen, das den gegenwärtigen Wochentag ausgibt. Ergänzen Sie die Lücke!

```
set '____________________'
case $1 in
Mo) echo Mondich;;
Di) echo Dienschdich;;
...
So) echo Sonndich;;
esac
```

Aufgabe 4

Wie kann man in einer einzigen Kommandozeile alle in einer Datei *x* enthaltenen Wörter in sortierter Reihenfolge ohne Duplikate erhalten?

Aufgabe 5

Jeder Eintrag in einem Verzeichnis besteht aus zwei Werten. Welche sind dies?

Aufgabe 6

Bei der Auflistung aller Einträge seines Heimatverzeichnisses stellt Prof. Fix erstaunt fest, dass es einen Eintrag „`...`“ gibt, also mit drei Punkten. Mit `ls ...` erhält er tatsächlich eine Auflistung des Verzeichnisses zwei Stufen höher. Anders als die Einträge „`.`“ und „`..`“ ist „`...`“ aber kein Verzeichnis, sondern ein Link. Jetzt fällt ihm auch wieder ein, dass er „`...`“ früher einmal eingeführt hatte. Wie wurde der Link erzeugt?

Aufgabe 7

Häufig legt man ein Verzeichnis an und wechselt sofort in dieses neue Verzeichnis. Der Versuch, dies mit einem Shellskript `mkdircd` zu lösen, dessen Inhalt `mkdir $1 && cd $1` lautet und ausführbar ist, scheitert aber, d.h. zwar wird ein neues Verzeichnis erzeugt, wenn es noch nicht existiert, man wechselt aber nicht dorthin.

(a) Warum?

(b) Wie kann man `mkdircd` aufrufen, damit es doch klappt?

Aufgabe 8

Starten Sie einen Hintergrundprozess *xyz*, der auch nach dem Abmelden weiterläuft!

Aufgabe 9

Richtig oder falsch: Die gleichzeitige Nutzung **eines** Textsegments durch **mehrere** Prozesse heißt auch *shared memory* und ist in UNIX Teil der Prozesssynchronisierung, zu der auch Semaphore, Streams und Sockets gehören. Kurze Begründung der Antwort.

Aufgabe 10

Woher weiß der `login`-Prozess, welche Shell er für den erfolgreich angemeldeten Benutzer starten soll?

Aufgabe 11

Warum braucht man in UNIX das SUID-Bit an Kommandos wie z.B. `passwd`? Das Kommando hat ja sowieso `root` als Besitzer, ist ausführbar für alle und `/etc/passwd` wiederum ist schreibgeschützt für alle außer `root`.

Aufgabe 12

Nehmen wir an, ein Benutzer möchte als Teil einer Abmelderoutine das momentane Arbeitsverzeichnis in eine versteckte Datei `.last_directory` im Heimatverzeichnis schreiben, um dieses Verzeichnis als Startpunkt beim nächsten Login zu lesen und dorthin zu wechseln. Wie lautet die Anweisung zum Schreiben in der Abmelderoutine?

_____ > _____

Aufgabe 13

Wo würde man den Teil zum Lesen von `.last_directory` und das Wechseln in das gespeicherte Verzeichnis aus Aufgabe 12 unterbringen, damit das beim Anmelden immer geschieht?

Aufgabe 14

Das Shellskript `smalldata` listet alle Einträge eines als Argument übergebenen Verzeichnisses mit ihrer Dateigröße auf, wenn die Datei weniger als 99 Zeichen enthält. Ergänzen Sie die Lücken!

```
if [ -d _________ ]
then
  cd _________
  set `ls`
  for file
  do
    if [ -f $file ]
    then
      size=`_____________________________ | wc -c`
      if [ "_______________" -lt "99" ]
      then echo _____________ has _____________ characters
      fi
    fi
  done
else echo _________ is not a directory
fi
```

Aufgabe 15

Beim Versuch, eine Datei *x* zu verdoppeln, indem man den Inhalt nochmals hinten anfügt, funktioniert die Pipeline `cat` *x* `| cat >>`*x*. Dagegen erhält man bei `cat` *x* `>>`*x* eine Fehlermeldung, die Eingabe- sei auch die Ausgabedatei. Erklären Sie das unterschiedliche Verhalten!

ENDE DER KLAUSUR

Prof. Dr. Lutz Wegner
Stefan Achler M.Sc. **26. Februar 2015**

Einführung in UNIX
Klausur zum Wintersemester 2014/15

MUSTERLÖSUNG

Name:.....Vorname:.....

Matr. Nr.:.....Studiengang:.....

Bearbeiten Sie alle Fragen! Hilfsmittel sind nicht zugelassen! Falls Sie für die Beantwortung der Aufgaben zusätzlichen Platz benötigen, verwenden Sie die Rückseiten.

Aufgabe	Punkte max.	Punkte erreicht
1	2	
2	2	
3	2	
4	2	
5	2	
6	2	
7	2 + 2	
8	2	
9	2	
10	2	
11	2	
12	2	
13	2	
14	4	
15	2	
Summe	**34**	

Aufgabe 1

Das Pseudogerät `/dev/null` wird gerne als „Biteimer“ bezeichnet, weil dorthin umgeleitete Daten geräuschlos verschwinden. Kann man aus `/dev/null` auch lesen und wenn ja, was erhält man dann?

Ja, man kann aus /dev/null lesen und erhält sofort EOF.

Aufgabe 2

Wo wir gerade bei `/dev/null` sind: Mit `ln /dev/null papierkorb` wollte sich Prof. Fix einen Link in seinem Heimatverzeichnis anlegen, was aber bei unserem System mit einer Fehlermeldung beantwortet wurde. Was besagt die Fehlermeldung sinngemäß?

Kein harter Link über Volumegrenzen (Partitions) hinweg.

Aufgabe 3

Für die längst überfällige Sprachanpassung ans Schwäbische haben wir das folgende Shellskript `heitisch` („heute-ist“) entworfen, das den gegenwärtigen Wochentag ausgibt. Ergänzen Sie die Lücke!

```
set '____date____'
case $1 in
Mo) echo Mondich;;
Di) echo Dienschdich;;
...
So) echo Sonndich;;
esac
```

Aufgabe 4

Wie kann man in einer einzigen Kommandozeile alle in einer Datei *x* enthaltenen Wörter in sortierter Reihenfolge ohne Duplikate erhalten?

cat x | tr ' ' '\n' | sort | uniq

Aufgabe 5

Jeder Eintrag in einem Verzeichnis besteht aus zwei Werten. Welche sind dies?

i-Nummer und Dateiname bzw. Verzeichnisname

Aufgabe 6

Bei der Auflistung aller Einträge seines Heimatverzeichnisses stellt Prof. Fix erstaunt fest, dass es einen Eintrag „`...`“ gibt, also mit drei Punkten. Mit `ls ...` erhält er tatsächlich eine Auflistung des Verzeichnisses zwei Stufen höher. Anders als die Einträge „`.`“ und „`..`“ ist „`...`“ aber kein Verzeichnis, sondern ein Link. Jetzt fällt ihm auch wieder ein, dass er „`...`“ früher einmal eingeführt hatte. Wie wurde der Link erzeugt?

ln -s ../.. ...

Aufgabe 7

Häufig legt man ein Verzeichnis an und wechselt sofort in dieses neue Verzeichnis. Der Versuch, dies mit einem Shellskript `mkdircd` zu lösen, dessen Inhalt `mkdir $1 && cd $1` lautet und ausführbar ist, scheitert aber, d.h. zwar wird ein neues Verzeichnis erzeugt, wenn es noch nicht existiert, man wechselt aber nicht dorthin.

(a) Warum?

Nach Rückkehr aus einer Untershell gilt wieder die alte Umgebung einschließlich des Arbeitsverzeichnisses

(b) Wie kann man `mkdircd` aufrufen, damit es doch klappt?

. ./mkdircd, auch richtig . mkdircd oder source mkdircd

Aufgabe 8

Starten Sie einen Hintergrundprozess *xyz*, der auch nach dem Abmelden weiterläuft!

nohup xyz &

Aufgabe 9

Richtig oder falsch: Die gleichzeitige Nutzung **eines** Textsegments durch **mehrere** Prozesse heißt auch *shared memory* und ist in UNIX Teil der Prozesssynchronisierung, zu der auch Semaphore, Streams und Sockets gehören. Kurze Begründung der Antwort.

Falsch, die gleichzeitige Nutzung eines read-only Textsegments hat nichts mit dem Prozesssynchronisierungskonzept shared memory zu tun.

Aufgabe 10

Woher weiß der `login`-Prozess, welche Shell er für den erfolgreich angemeldeten Benutzer starten soll?

Die login-Shell wird für jeden Teilnehmer in der /etc/passwd-Datei festgehalten.

Aufgabe 11

Warum braucht man in UNIX das SUID-Bit an Kommandos wie z.B. `passwd`? Das Kommando hat ja sowieso `root` als Besitzer, ist ausführbar für alle und `/etc/passwd` wiederum ist schreibgeschützt für alle außer `root`.

Durch das SUID-Bit läuft der Prozess mit den Rechten des Besitzers der Datei, also root.

Aufgabe 12

Nehmen wir an, ein Benutzer möchte als Teil einer Abmelderoutine das momentane Arbeitsverzeichnis in eine versteckte Datei `.last_directory` im Heimatverzeichnis schreiben, um dieses Verzeichnis als Startpunkt beim nächsten Login zu lesen und dorthin zu wechseln. Wie lautet die Anweisung zum Schreiben in der Abmelderoutine?

___pwd____ > ___$HOME/.last_directory____

Aufgabe 13

Wo würde man den Teil zum Lesen von `.last_directory` und das Wechseln in das gespeicherte Verzeichnis aus Aufgabe 12 unterbringen, damit das beim Anmelden immer geschieht?

in .profile oder ähnlicher beim Start ausgeführter Datei

Aufgabe 14

Das Shellskript `smalldata` listet alle Einträge eines als Argument übergebenen Verzeichnisses mit ihrer Dateigröße auf, wenn die Datei weniger als 99 Zeichen enthält. Ergänzen Sie die Lücken!

```
if [ -d ___$1___ ]
then
  cd ___$1___
  set `ls`
  for file
  do
    if [ -f $file ]
    then
      size=`____cat $file___ | wc -c`
      if [ "___$size__" -lt "99" ]
      then echo ___$file___ has ___$size___ characters
      fi
    fi
  done
else echo __$1__ is not a directory
fi
```

Aufgabe 15

Beim Versuch, eine Datei *x* zu verdoppeln, indem man den Inhalt nochmals hinten anfügt, funktioniert die Pipeline `cat` *x* `| cat >>`*x*. Dagegen erhält man bei `cat` *x* `>>`*x* eine Fehlermeldung, die Eingabe- sei auch die Ausgabedatei. Erklären Sie das unterschiedliche Verhalten!

Die Shell achtet bei Umlenkungen nicht darauf, ob eine Datei x zugleich Quelle und Ziel ist. Das cat-Kommando unter Linux merkt dies aber und gibt eine Fehlermeldung aus.

ENDE DER KLAUSUR