Kurze Einführung in Linux

Martin Eichner
Institut für Medizinische Biometrie
Universität Tübingen

21. Januar 2002

Inhaltsverzeichnis

— — — — — — — — **Einheit 7** — — — — — — — —

— — — — — — — — **Einheit 8** — — — — — — — —

— — — — — — — — **Einheit 9** — — — — — — — —

— — — — — — — — **Einheit 10** — — — — — — — —

— — — — — — — — **Einheit 11** — — — — — — — —

— — — — — — — — **Einheit 12** — — — — — — — —

— — — — — — — — **Einheit 13** — — — — — — — —

— — — — — — — — **Einheit 14** — — — — — — — —

— — — — — — — — **Einheit 15** — — — — — — — —

— — — — — — — — **Einheit 16** — — — — — — — —

— — — — — — — — **Einheit 17** — — — — — — — —

Teil
Vorwort

1 Persönliches

Eigentlich wollte ich schon lange einmal lernen, mit Linux umzugehen ... Meist fehlt einem aber die Energie, sich hinzusetzen und sich da durchzuwuseln. Als wir vor einiger Zeit damit anfingen, uns am Institut gegenseitig zu unterrichten, kam mir die Idee, einen Linux-Kurs zu geben. Nicht, dass so etwas nicht schon irgendwo angeboten würde, aber schließlich lernt man am besten, was man selbst unterrichtet. Ich war den anderen zwar meist nur eine Nasenlänge voraus, aber was soll's – hauptsache es macht Spaß und es kommt 'was dabei 'raus.

2 Über diesen Kurs

Diesen Kurs habe ich für Leute gemacht, die – wie ich – schon einmal etwas mit Linux und/oder anderen Betriebssystemen herumgespielt haben, und aus ihrem Computer mehr herausholen möchten. Ich gehe dabei extrem pragmatisch und persönlich vor: Kommandos und Kommando-Optionen, die ich im Alltag verwenden möchte, bespreche ich, alles andere wird konsequent weggelassen. Theorie wird nur besprochen, wenn sie hilft, mit konkreten Aufgaben besser klar zu kommen. Gelernt wird nur an Beispielen. Am meisten werden diejenigen profitieren, die sich mit einer kindlichen Freude am Spiel ihre eigenen Beispiele ausdenken, vorgeschlagene Optionen und Kommandos kombinieren, und vielleicht auch 'mal ganz bösartig versuchen, durch gewagte Konstellationen den Computer in die Knie zu zwingen.

3 Zu Risiken und Nebenwirkungen ...

Meine Auffassung über das, was der Computer tut, und meine Namensgebungen sind größtenteils "auf meinem eigenen Mist" gewachsen. Um gesundheitliche Schäden zu vermeiden, sollten echte Linux-Insider die Lektüre dieses Skriptes besser meiden. Wer es versucht, mit diesem Kurs "Linux zu lernen", wird nicht nur die manchmal etwas steile Lernkurve zu meistern haben, sondern wird sich auch an den von mir gemachten Fehlern erfreuen (soweit sie meine Kollegen noch nicht ausgemerzt haben) und wird auch meine Art Humor er- und vertragen müssen. In diesem Sinne: viel Spaß!

Teil I

Arbeiten mit Dateien und Verzeichnissen — Basics

Einheit 1

1 Was liegt da alles 'rum? – "ls", "du" and "df"

Das erste, was man auf der Kommandozeile benötigt, ist eine gewisse Orientierung, dh. man sollte zumindest wissen, welche Dateien überhaupt im aktuellen Verzeichnis vorhanden sind.

ls	gibt alle Dateien des aktuellen Verzeichnisses, die nicht mit einem Punkt beginnen, (in Kurzform) an.

Dabei lässt sich auch gezielt nach besonderen Dateien suchen, indem man z. B. Wildcards einsetzt: "*" steht für Text von beliebiger Länge. "?" steht für ein einzelnes Zeichen. "[a-z]" steht für einen beliebigen Kleinbuchstaben und "[a-zA-Z]" steht für einen beliebigen Buchstaben (einschließlich der Umlaute "ä", "Ä", "ö", "Ö", "ü" und "Ü", sowie des deutschen "ß"). "[0-9]" steht für eine beliebige Ziffer im Bereich von "0" bis "9". Alternative Zeichenketten können gesucht werden, indem sie von geschweiften Klammern umschlossen werden: "{text1,text2,text3}". Wenn Verzeichnisse gefunden werden, auf welche die angegebene Beschreibung zutrifft, wird deren kompletter Inhalt angezeigt.

ls *t*en*	sucht nach Dateien, die mit "t" beginnen und auf "en" enden.
ls *data?.txt*	sucht nach Dateien, bei denen "?" durch ein beliebiges Zeichen (Ziffer, Buchstabe oder ein anderes erlaubtes Zeichen) ersetzt ist.
ls **.{txt,dat,jmp}*	zeigt alle Dateien an, die entweder mit ".txt", mit ".dat" oder mit ".jmp" enden.
ls *[a-c][1-2].txt*	sucht nach den Dateien "a1.txt", "a2.txt", "b1.txt", "b2.txt", "c1.txt", "c2.txt".

Diese Bereichsangabe ist ein extrem mächtiges Werkzeug. Man kann innerhalb der eckigen Klammern auch eine Aufzählung (ohne Komma etc.) angeben und/oder mehrere Bereiche kombinieren:

ls *[a-c1-3].txt*	sucht nach den Dateien "a.txt", "b.txt", "c.txt", "1.txt", "2.txt" und "3.txt".
ls *[abzABZ].txt*	sucht nach den Dateien, die mit klein oder groß geschriebenem "a", "b" oder "z" beginnen.

Es kommt noch schlimmer: Ein solcher Bereich kann durch ein vorausgehendes "^" verneint werden: z. B. bedeutet [^a-c], dass keiner der Buchstaben "a", "b" oder "c" vorkommen darf.

ls *[^A]**	sucht nach allen Dateien, die nicht mit "A" beginnen.
ls **[^0-9]*	sucht nach allen Dateien, die nicht auf eine Ziffer enden.
ls *[^a-z0-9]**	sucht nach allen Dateien, die weder mit einem Kleinbuchstaben noch mit einer Ziffer beginnen (das ist nicht notwendigerweise identisch mit "**ls** *[A-Z]**").

Natürlich kann man auch den Pfad angeben, für den man sich die vorhandenen Dateien anzeigen lassen möchte. Die Pfadangabe kann entweder absolut sein oder relativ zur momentanen Position:

ls */home/eichner/work*	zeigt alle Dateien im angegebenen Verzeichnis an. Da der Pfad mit dem Zeichen "/" beginnt, handelt es sich um eine absolute Pfadangabe, die immer gilt, egal in welchem Verzeichnis man sich gerade befindet.
ls *~/work*	dient als Abkürzung für voriges Kommando, vorausgesetzt, dass "/home/eichner" das home directory des Benutzers ist.
ls *work*	zeigt die gleichen Dateien an, wenn man sich bereits im Verzeichnis "/home/eichner" befindet. Das Fehlen des initialen "/" zeigt, dass es sich um eine relative Pfadangabe handelt.
ls ..	zeigt alle Dateien im übergeordneten Verzeichnis an.

Wenn man einige Dateien nicht angezeigt bekommen möchte, benutzt man:

ls *-I∗.txt*	zeigt alle Dateien außer denen, die mit ".txt" enden, an. **Vorsicht:** Zwischen "-I" und dem nachfolgenden Muster darf kein Leerzeichen stehen ("I" steht für "ignore-pattern").

Den Inhalt aller Unterverzeichnisse und deren Unterverzeichnisse gleichzeitig erhält man durch

ls -R	zeigt außer dem aktuellen Verzeichnis auch den Inhalt aller Unterverzeichnisse an ("R" steht für "recursive").

Um an die sonst nicht angezeigten Dateien heranzukommen, die mit einem Punkt beginnen, benötigt man das Kommando

ls -a	zeigt alle Dateien an, auch diejenigen, welche mit einem Punkt beginnen ("a" steht für "all").
la	mögliche Abkürzung für das vorige Kommando.

Für den Fall, dass man sich auch noch für die Größe der angezeigten Dateien und/oder deren Benutzerrechte interessiert, gibt man besser das folgende Kommando ein. Durch die zusätzliche Option "h" kann man sich vom Computer das lästige Abzählen der Stellen bei Dateien sparen, die mehrere Megabytes groß sind:

ls -l	Alle Dateien und Unterverzeichnisse werden im ausführlichen Format angezeigt ("l" steht für "long").
ll	mögliche Abkürzung für das vorige Kommando.
ls -lh	Alle Dateien und Unterverzeichnisse werden im ausführlichen Format angezeigt; die Dateigröße wird in eine angenehm lesbare Form gebracht ("l" steht für "long", "h" für "human-readable").

Nede Zeile beginnt mit einer Buchstabenkombination, wie "drwxrwxrwx", in der jeder Buchstabe auch durch ein "-" ersetzt worden sein kann. Steht an erster Stelle das "d" (directory), so handelt es sich um ein Verzeichnis. Danach kommen drei Blöcke, in denen "rwx" die Rechte für den Benutzer, dessen Gruppe und für die Allgemeinheit festlegen: "r" (read) steht für das Recht, die Datei zu lesen, "w" (write) für das Recht, die Datei zu verändern oder zu löschen und "x" (execute) für

das Recht, die Datei auszuführen (falls möglich). Die letzte Spalte vor Erstellungs-Datum und -Uhrzeit gibt die Größe der Datei in Bytes an.

Eine Zusammenfassung der Speicherbelegung derhält man durch das Kommando

du	zeigt den Speicherbedarf der Dateien im aktuellen Verzeichnis sowie von jedem Unterverzeichnis (rekursiv) an. Die Summe der Speicherbelegungen steht in der letzten Zeile ("**du**" steht für "disk usage").
du -sh	zeigt nur eine Zusammenfassung der Speicherbelegung an und verzichtet auf Einzelheiten ("s" steht für "summary"; "h" steht für "human-readable").
du -sh *Dat*	zeigt nur eine Zusammenfassung der Speicherbelegung im Unterverzeichnis "Dat" (Wildcards und Bereichsangaben sind ebenso möglich wie beim ls-Kommando).

Wenn man jetzt noch wissen möchte, wieviel Platz noch nicht verbraten ist, benötigt man das Kommando

df -h	zeigt für jeden "gemounteten" Datenträger an, wieviel Platz belegt bzw. noch frei ist ("**df**" steht für "disk free"; "h" steht für "human-readable").

Aufgaben:

Sie befinden sich im Verzeichnis "/imbNet/pool/LinuxKurs/A", "/imbNet/pool/LinuxKurs/B" oder "/imbNet/pool/LinuxKurs/C". Finden Sie die eleganteste Lösung der folgenden Aufgaben, ohne dazu das Verzeichnis zu wechseln. Die besten Lösungen erfordern i. d. R. die Kombination der oben angegebenen Beispiele, also einen kräftigen Schuss Experimentierfreudigkeit und Transfervermögen.

I. 1.1. Listen Sie *alle* Dateien des Unterverzeichnisses "Benutzer" auf.

I. 1.2. Listen Sie die Dateien des übergeordneten Verzeichnisses auf. Benutzen Sie dafür das ausführliche Format.

I. 1.3. Listen Sie die Dateien auf, die mit ".txt" enden.

I. 1.4. Listen Sie die Dateien auf, welche die Zeichenkette "dat" enthalten.

I. 1.5. Listen Sie die Dateien auf, die mit "a" beginnen.

I. 1.6. Listen Sie die Dateien auf, die nicht mit einem Buchstaben beginnen.

I. 1.7. Listen Sie die Dateien auf, welche eine Ziffer zwischen "0" und "5" enthalten.

I. 1.8. Die harte Nuss:
Listen Sie die Dateien auf, welche keine der Ziffern zwischen "0" und "5" enthalten.

I. 1.9. Wer (Benutzer, Gruppe, Allgemeinheit) darf die Datei "Xdat1.txt" lesen? Wer darf sie verändern?

I. 1.10. Wieviel Platz wurde *insgesamt* für den LinuxKurs auf dem Pool belegt? Bitte lassen Sie sich das Ergebnis in einer angenehm lesbaren Form anzeigen.

I. 1.11. Wieviel Platz ist auf dem Pool noch verfügbar?

Lösungen:

I. 1.1. **la** *Benutzer*

I. 1.2. **ll** *..*

I. 1.3. **ls** *∗.txt*

I. 1.4. **ls** *∗dat∗*

I. 1.5. **ls** *a∗*

I. 1.6. **ls** *[^a-zA-Z]∗*

I. 1.7. **ls** *∗[0-5]∗*

I. 1.8. **ls -l** *∗[0-5]∗*

I. 1.9. **ll**
lesen: Benutzer, Gruppe, Allgemeinheit
schreiben: nur Benutzer

I. 1.10. **du -sh** *../../LinuxKurs*

I. 1.11. **df -h**

2 Kopf oder Schwanz – "head" or "tail"

Einheit 2

"**head**" und "**tail**" geben die Anfangs- und Endzeilen einer Textdatei auf den Bildschirm aus. Diese Programme können auch mit den Wildcards (∗?) und mit Bereichsangaben [ ] kombiniert werden.

head *dat.txt*	gibt die ersten 10 Zeilen (Voreinstellung) der Datei "dat.txt" auf den Bildschirm aus.
head *-12 dat.txt*	gibt die ersten 12 Zeilen der Datei "dat.txt" auf den Bildschirm aus.
tail *dat.txt*	gibt die letzten 10 Zeilen (Voreinstellung) der Datei "dat.txt" auf den Bildschirm aus.
tail *-2 ∗.txt*	gibt die letzten 2 Zeilen aller Dateien auf, die auf ".txt" enden.
tail -f *dat.txt*	gibt die letzten 10 Zeilen der Datei "dat.txt" auf den Bildschirm aus. Sobald die Datei verändert wird, werden die jeweils aktuellen 10 Endzeilen ausgegeben. Beendet wird das Programm durch "Crtl C" ("f" steht für "follow").
tail *-5***f** *dat.txt*	wie der vorige Befehl; es werden aber nur 5 Zeilen ausgegeben.

Mit der letzten Option kann man den sehr gut den Fortschritt eines Rechenprogramms verfolgen, das seinen Output in eine Textdatei schreibt.

3 Mehr oder weniger – "more" or "less"

Zum schnellen Auflisten von Textdateien auf die Kommandozeile gibt es die Befehle "**more**" und "**less**" (wenn Sie "q" eingeben, verlassen Sie das Programm wieder).

less *demo.txt*	die Textdatei "demo.txt" wird auf dem Bildschirm angezeigt. Um auf die Kommandozeile zurückzukehren, geben Sie "q" ein.

Manchmal ist es nützlich, gleich noch die Zeilennummer zu erfahren, in der eine bestimmte Information steht:

less -N *demo.txt*	jede Zeile wird numeriert auf den Bildschirm ausgegeben, sonst wie voriges Kommando ("N" steht für "number").

Recht angenehm ist auch die Möglichkeit, "**less**" zum Suchen von Textbegriffen einzusetzen. Da es sich beim einzugebenden Suchbegriff um eine "regular expression" handelt (näheres hierzu siehe Teil II, Abschnitt 2), lassen sich die üblichen Bereichsangaben [ ] verwenden. Die Wildcards (∗?) funktionieren hier leider nicht in der gewohnten Weise. Anstelle des Wildcardzeichens "?" kann man jedoch den Punkt "." verwenden.

less -p *poly demo.txt*	alle Vorkommnisse des nach "-p" eingegebenen Suchmusteres werden im Text markiert; das Programm springt automatisch zu der Zeile, in welcher der Suchbegriff "poly" zum ersten mal vorkommt ("p" steht für "pattern").

less -p *"for poly" demo.txt*	ähnlich wie voriges Kommando; die Verwendung von Leerzeichen und Sonderzeichen innerhalb des Suchbegriffs kann jedoch zu Zweideutigkeiten führen (ohne Anführungsstriche hätte der Computer "for" als Suchbegriff verstanden und in der Datei "poly" danach gesucht).
less -p *for\ poly demo.txt*	identisch mit vorigem Kommando; statt den Suchbegriff in Anführungszeichen einzuschließen kann ein Leerzeichen durch ein vorausgehendes "\" gekennzeichnet werden.
less -p *"0,[0-9]0 DM" demo.txt*	Beispiel für die Verwendung von Bereichsangaben in Suchbegriffen: es wird nach allen Vorkommen der Zeichenketten von "0,00 DM" bis "0,90 DM" im Text gesucht.
less -p *19.0 demo.txt*	**Vorsicht:** Der Punkt "." wird in "regular expressions" wie das Wildcardzeichen "?" verwendet. Es wird also nach der Folge "19 – anderes Zeichen – 0" gesucht (z. B. "1990"), nicht nur nach "2.0". **Anmerkung:** Nach "19.0" kann man suchen, wenn man den Punkt innerhalb einer Bereichsangabe versteckt: "**less -p** *19[.]0*".

Sobald man die Bildschirmanzeige vor sich hat, kann man mit den Cursorbewegungstasten [↑] und [↓] sowie mit den [Bild↑]– und [Bild↓]–Tasten innerhalb des Textes vor- und zurückblättern. Lassen Sie sich die Datei "dat.txt" auf dem Bildschirm anzeigen und probieren Sie die folgenden Möglichkeiten innerhalb der "**less**"-Anzeige aus:

50	springt die angegebene Anzahl von Zeilen (hier: "50") vor.
/G141	sucht (von der gegenwärtigen Position vorwärts) nach dem angegebenen Suchmuster (hier: "G141"; Details siehe oben); die Suche wird mit der zweiten angezeigten Zeile begonnen.
/![0-9]0	sucht nach Zeilen, in denen der Suchbegriff *nicht* vorkommt. Alle Vorkommnisse des Suchbegriffes werden markiert und – falls Zeilen ohne den Suchbegriff existieren – wird die erste Zeile, in welcher der Suchbegriff nicht vorkommt, auf dem Bildschirm angezeigt; die Suche wird mit der zweiten angezeigten Zeile begonnen
n	wiederholt den letzten Suchbefehl.
?*murks*	sucht (von der gegenwärtigen Position rückwärts) nach dem angegebenen Suchmuster (hier: "murks").
q	beendet die Anzeige der Datei.

Weitere Suchoptionen siehe Teil III, Abschnitt **??**. Das Programm "**more**" ähnelt in weiten Zügen dem Programm "**less**", ist aber in den Merkmalen, die mir wichtig und brauchbar erschienen genau das Gegenteil von dem, was der Name vorgibt: Die meisten der oben beschriebenen Optionen von "**less**" funktionieren nicht (oder zumindest anders). Aber trotzdem; warum nicht:

more *dat.txt*	zeigt den Inhalt der Datei "dat.txt" auf dem Bildschirm an.

4 Alles für die Katz – "cat" and "tac"

Mal eben schnell eine Datei ansehen? Eine neue Datei anlegen oder eine bestehende Datei kopieren? Oder mehrere Dateien miteinander verknüpfen? Was auch immer das Problem sei. Die Antwort auf alles heißt "**cat**". Na ja, zumindest auf recht vieles.

cat *hallo.txt*	gibt die Datei "hallo.txt" auf den Bildschirm aus.
cat *hallo.txt* > *hi*	die Ausgabe wird in "hi" umgelenkt; d. h. "hallo.txt" wird kopiert und als "hi" abgespeichert.
cat *[0-9].txt* > *neu*	die Dateien "0.txt" bis "9.txt" werden (soweit vorhanden) miteinander verbunden und als Datei "neu" abgespeichert.
cat *[0-9].txt* ≫ *neu*	die Dateien "0.txt" bis "9.txt" werden (soweit vorhanden) ans Ende der Datei "neu" angehängt. Im Gegensatz zum vorigen Befehl wird "neu" am Anfang nicht gellöscht (falls vorhanden) sondern nur am Ende weitergeshrieben.
cat > *memo*	Nach Eingabe dieses Kommandos wartet der Computer auf weitere Texteingabe. Man hat die Möglichkeit beliebig viele Zeilen Text zu schreiben. Wenn man fertig ist, schließt man die letzte Zeile noch mit Return ab und gibt anschließend "Crtl Z" oder "Crtl C" ein. Der eingegebene Text wird unter dem Namen "memo" gespeichert.
cat ≫*memo*	Ein paar Zeilen bei der vorigen Eingabe in die Datei "memo" vergessen? Macht nichts, dieser Befehl hängt sie an's Ende der Datei an (Bedienung wie voriger Befehl).

Wer Lust hat, Dinge auf den Kopf zu stellen, für den ist das Programm "**tac**" ("**cat**" rückwärts geschrieben) genau das richtige:

tac *memo*	gibt die Datei "memo" beginnend mit der letzten Zeile auf den Bildschirm aus.
tac *memo* > *omem*	Schreibt die Datei "memo" beginnend mit der letzten Zeile in die Datei "omem".

Aufgaben:

Die folgenden Aufgaben sollen nur mit den Befehlen "**head**", "**tail**", "**cat**" und "**less**" gelöst werden.

I. 4.1. Geben Sie die erste Zeile jeder Datei, die mit ".txt" endet, aus.

I. 4.2. Wie sehen die letzten 3 Zeilen der Datei "brief.txt" aus?

I. 4.3. Legen Sie eine Datei namens "neu.txt" an, die nichts anderes als die Worte "viele Grüße" enthält

I. 4.4. Legen Sie eine Kopie der Datei "neu.txt" unter dem Namen "neu2.txt" an.

I. 4.5. Hängen Sie die Datei "neu.txt" an die bestehende Datei "brief.txt" hinten an.

I. 4.6. Welches ist das *einfachste* Kommando, um alle Dateien, deren Namen mindestens eine Ziffer enthalten, zur Datei "Ziffer.txt" zusammenzufassen?

I. 4.7. In welcher Zeile der Datei "dat.txt" kommt der Eintrag "G142" zum ersten mal vor?

Lösungen:

I. 4.1. **head** *-1 ∗.txt*

I. 4.2. **tail** *-3 brief.txt*

I. 4.3. **cat** > *neu.txt* Return
viele Grüße Return
Crtl Z

I. 4.4. **cat** *neu.txt* > *neu2.txt*

I. 4.5. **cat** *neu.txt* >> *brief.txt*

I. 4.6. **cat** *∗[0-9]∗* > *Ziffer.txt*

I. 4.7. **less -N -p** *G142 dat.txt*

Einheit 3

5 Dateien kopieren – "cp"

Dateien kopiert man mit dem Kommando "**cp**". Vor dem Namen der zu kopierenden Datei und/oder vor den Namen der Zieldatei können wieder (relative oder absolute) Pfadangaben stehen. Die üblichen Wildcards (∗?) und Bereichsangaben [ ] sind beim Kopieren prinzipiell erlaubt, sind aber nicht immer sinnvoll und führen auch dort, wo sie sinnvoll erscheinen nicht immer zum erwünschten Resultat.

cp *dat1 Z*	legt eine Kopie von Datei "dat1" unter dem Namen "Z" ab ("**cp**" steht für "copy"). Falls bereits ein Verzeichnis des Namens 'Z' existiert, wird die Datei "dat1" dorthin kopiert und die Kopie trägt den Namen "dat1".
cp *dat? Dat?*	**NEIN!** Von der Verwendung von Wildcards ist dringend abzuraten, wenn Dateien mit einem neuen Namen (hier: "Dat?") abgespeichert werden sollen.
cp *∗.txt x/*	kopiert alle Dateien, die mit ".txt" enden, in das Verzeichnis "x". **Vorsicht:** Sollten dort Dateien gleichen Namens existieren, werden diese ohne Warnung überschrieben.
cp -i *[Dd]at∗ ..*	kopiert alle Dateien, die mit "dat" oder "Dat" beginnen, in das übergeordnete Verzeichnis. Falls dort eine gleichnamige Datei existiert, fragt der Computer nach, bevor diese Datei überschrieben wird ("i" steht für "interactive").
cp -u *[a-zA-Z]∗ ..*	kopiert alle Dateien, die mit einem Buchstaben beginnen, in das übergeordnete Verzeichnis. Falls dort eine gleichnamige Datei existiert, wird diese nur überschrieben, wenn sie älter ist als die zu kopierende Datei ("u" steht für "update").
cp -v *∗[0-9]∗ x/*	kopiert alle Dateien, die eine Ziffer enthalten, in das Verzeichnis "x". Alle Kopieraktionen werden auf der Kommandozeile aufgelistet ("v" steht für "verbose").

6 Dateien umbenennen und verschieben – "mv"

Zum Umbenennen und zum Verschieben von Dateien (und Verzeichnissen) benutzt man unter Linux das gleiche Kommando "**mv**". Die üblichen Wildcards (∗?) und Bereichsangaben [ ] sind beim "**mv**"-Kommando prinzipiell erlaubt, sind aber – ebenso wie beim Kopieren – nicht immer sinnvoll und führen auch dort, wo sie sinnvoll erscheinen nicht immer zum erwünschten Resultat. Zunächst eine kurze Übersicht über die möglichen Resultate des "**mv**"-Kommandos:

Quelle	Ziel	Aktion
Datei	nicht bestehende Datei	Datei umbenennen
Datei	bestehende Datei	bestehende Datei überschreiben
Datei(en)	Verzeichnis	Datei(en) ins Verzeichnis kopieren
Verzeichnis	nicht bestehendes Verz.	Verzeichnis umbenennen
Verzeichnis(se)	bestehendes Verzeichnis	Verzeichnis(se) als Unterverzeichnis(se) in das bestehende Verzeichnis verschieben

Das Umbenennen und Verschieben von Verzeichnissen wird in der Einheit über Verzeichnisse ausführlich besprochen.

mv *dat1 Z*	benennt die Datei "dat1" in "Z" um. **Vorsicht:** Falls bereits eine Datei namens "Z" existiert, wird diese ohne Warnung gelöscht. Falls dagegen ein Unterverzeichnis namens "Z" existiert, wird die Datei "dat1" in dieses Verzeichnis verschoben und behält ihren ursprünglichen Namen ("**mv**" steht für "move").
mv *dat? Dat?*	**NEIN!** Von der Verwendung von Wildcards ist dringend abzuraten, wenn Dateien einen neuen Namen (hier: "Dat?") bekommen sollen.
mv -i *dat* ..*	verschiebt alle Dateien, die mit "dat" beginnen, in das übergeordnete Verzeichnis. Falls dort eine gleichnamige Datei existiert, fragt der Computer nach, bevor diese Datei überschrieben wird ("i" steht für "interactive").
mv -u *d* ..*	verschiebt alle Dateien, die mit "d" beginnen, in das übergeordnete Verzeichnis. Falls dort eine gleichnamige Datei existiert, wird diese nur überschrieben, wenn sie älter ist als die zu kopierende Datei ("u" steht für "update").
mv -v **[0-9]* x/*	verschiebt alle Dateien, die eine Ziffer enthalten, in das Verzeichnis "x". Alle Aktionen werden auf der Kommandozeile aufgelistet ("v" steht für "verbose").

7 Dateien löschen – "rm"

Vorsicht beim Löschen von Dateien unter Linux. Es gibt bis heute keinen (unkomplizierten) Weg, irrtümlich gelöschte Dateien wieder zum Leben zu erwecken. Besonders interessant wird es, wenn man beim Löschen die üblichen Wildcards (*?) und Bereichsangaben [ ] verwendet und dazu noch Optionen angibt, die dem Computer jedes Rückfragen verbieten und möglichst gleich die Unterverzeichnisse mit platt machen (siehe die Einheit über Verzeichnisse).

rm *dat**	löscht alle Dateien im aktuellen Verzeichnis, die mit "dat" beginnen ("**rm**" steht für 'remove").
rm -i *[a-c]**	löscht alle Dateien, die mit "a", "b" oder "c" beginnen, fragt aber vor jedem Löschen nach ("i" steht für "interactive").
rm -v **.txt*	löscht alle Dateien, die mit ".txt" enden und protokolliert jede Aktion auf dem Bildschirm ("v" steht für "verbose").

Aufgaben:

Lösen Sie alle Aufgaben lediglich unter Verwendung der Befehle "**cp**", "**mv**" und "**rm**" ohne weitere Hilfsmittel.

I. 7.1. Legen Sie eine Kopie von "brief.txt" im Unterverzeichnis "x" an.

I. 7.2. Kopieren Sie alle Dateien Ihres Verzeichnisses "/imbNet/pool/LinuxKurs/A" (bzw. "B" oder "C") in das entsprechende Verzeichnis "/imbNet/pool/Eichner/LinuxKurs/A" (bzw. "B" oder "C"). Verfolgen Sie den Kopiervorgang am Bildschirm und beachten Sie, dass nur ältere Dateien überschrieben werden sollen.

I. 7.3. Verschieben Sie alle Dateien, deren Namen weder auf einen Buchstaben noch auf eine Ziffer enden, in das Unterverzeichnis "abstrus".

I. 7.4. Benennen Sie die Datei "brief.txt" um "AlterBrief.txt"

I. 7.5. Löschen Sie alle Dateien, deren Namen mit einer Ziffer beginnen.

I. 7.6. Löschen Sie alle Dateien des Unterverzeichnisses "x"

Lösungen:

I. 7.1. **cp** *brief.txt x*

I. 7.2. **cp -uv** * */imbNet/pool/Eichner/LinuxKurs/A*

I. 7.3. **mv** **[^A-Za-z0-9] abstrus*

I. 7.4. **mv** *brief.txt AlterBrief.txt*

I. 7.5. **rm** *[0-9]**

I. 7.6. **rm** *x/**

Einheit 4

8 Zwischen Verzeichnissen hin und her wechseln – "cd", "pwd", "pushd" and "popd"

Wenn man nicht immer nur im home directory sitzen möchte, wird man sich wohl oder übel von dort fortbewegen müssen. Dazu gibt's das "**cd**"-Kommando.

cd	wird "**cd**" ohne Argumente eingegeben, so wechselt man in das home directory. Für Benutzer "eichner" ist das Kommando also gleichbedeutend mit "cd /home/eichner" bzw. mit "**cd** ~" ("**cd**" steht für "change directory").
cd ..	wechselt in das übergeordnete Verzeichnis.
cd /	wechselt in das oberste Verzeichnis.
cd -	macht den *zuletzt* durchgeführten Verzeichniswechsel rückgängig. Durch wiederholte Eingabe von "**cd** -" springt man zwischen zwei Verzeichnissen hin und her.

In der Regel wird man genaue Vorstellungen haben, zu welchem Pfad man wechseln will. Die Pfadangabe kann wieder relativ zum momentanen Verzeichnis oder absolut erfolgen:

cd */imbNet/pool*	wechselt in das Verzeichnis "/imbNet/pool" (absolute Pfadangabe).
cd *LinuxKurs/A*	wechselt in das Unterverzeichnis "A" des Unterverzeichnisses "LinuxKurs" (relative Pfadangabe).

Wenn Sie Fan von Wildcards sind, können Sie einmal ausprobieren, wie Ihr Computer auf Befehle wie "`cd D*`" oder "`cd ?rief`" reagiert. Da die Resultate zu viel Nachdenken erfordern, würde ich jedoch von der Verwendung der Wildcards bei "**cd**" abraten.
Oft ist es mir zu mühsam, lange Pfadnamen einzugeben. Eine Möglichkeit, Pfade zu speichern, ist gegeben durch das Kommando

pushd .	speichert das momentane Verzeichnis (".") auf dem "Stack" ("**pushd**" steht für "push directory").

Zu dem durch "**pushd**" gespeicherten Verzeichnis kann man später durch Eingabe des folgenden Kommandos zurückspringen:

popd	springt zu dem zuletzt mit "**pushd**" eingegebenen Verzeichnis und löscht den "**pushd**"-Eintrag. Wurden mehrere "**pushd**"-Kommandos eingegeben, so werden diese – beginnend mit dem letzten – abgearbeitet ("**popd**" steht für "pop directory").

Jetzt noch der Befehl für diejenigen, die überhaupt nicht mehr wissen, wo sie sich befinden:

pwd	gibt eine (ausführliche) Beschreibung des aktuellen Verzeichnisses ("**pwd**" steht für "print working directory").

9 Verzeichnisse anlegen, bearbeiten und löschen – "mkdir" and "rmdir"

Verzeichnisse werden unter Linux im großen und ganzen ebenso behandelt wie normale Dateien, so dass fast alles, was zuvor über Dateien gesagt wurde auch hier angewandt werden kann. Bei allen Verzeichnisbearbeitungen können die Wildcards (*?) und die Bereichsangaben [ ] im Dateinamen benutzt werden, soweit dies sinnvoll ist. Bitte wechseln Sie in Ihr Arbeitsverzeichnis "/imbNet/pool/LinuxKurs/A" (bzw. "B" oder "C") bevor Sie die folgenden Beispiele an der Konsole nachvollziehen.

mkdir *brief*	legt ein Verzeichnis namens "brief" im aktuellen Verzeichnis ab ("**mkdir**" steht für "make directory").
md *brief*	mögliche Abkürzung des vorigen Kommandos.
mkdir -p *Br/1/sik*	legt das Verzeichnis "sik" unter "Br/1" an; falls die Verzeichnisse "1" und "Br" noch nicht existieren, werden diese ebenfalls angelegt ("p" steht für "parent").
mkdir *Verz[1-3]*	**NEIN!** Multiple Verzeichnisse kann man so leider nicht anlegen.

Verzeichnisse kopiert oder verschieben man durch

cp -r *Briefe sik*	falls noch kein Verzeichnis "sik" existiert, erstellt dieser Befehl eine komplette Kopie des Verzeichnisses "Brief" unter dem Namen "sik" (einschließlich aller Unterverzeichnisse und der darin enthaltenen Dateien). Falls "sik" bereits besteht, wird ein Unterverzeichnis "Briefe" innerhalb von "sik" angelegt und alle Dateien und Unterverzeichnisse werden dorthin kopiert ("r" steht für "recursive").
mv *brief OldBr*	falls nicht schon ein Verzeichnis "OldBr" existiert, benennt dieser Befehl das Verzeichnis "brief" in "OldBr" um; andernfalls wird "brief" verschoben und als Unterverzeichnis von "OldBr" angelegt. Dabei verschiebt der Befehl "**mv**" *per default* sämtliche Unterverzeichnisse von "brief" mit.

Verzeichnisse löscht man durch

rmdir *OldBr*	löscht das Verzeichnis "OldBr", falls dies völlig leer ist, d. h. weder Dateien noch Unterverzeichnisse enthält ("**rmdir**" steht für "remove directory").
rmdir *V[1-3]*	löscht die Verzeichnisse "V1", "V2" und "V3", falls diese leer sind.
rmdir -p *Br/1/sik*	durch die Angabe "p" versucht Linux, möglichst den ganzen Verzeichnisbaum "Br/1/sik" zu löschen. Beginnend mit dem letzten Verzeichnis "sik" wird geprüft, ob das Verzeichnis leer ist. Wenn ja, wird es gelöscht und das übergeordnete Verzeichnis geprüft und evtl. gelöscht ("p" steht für "parent").

rm -rf *Schrott* — **Vorsicht:** Dies ist das absolute Killer-Kommando: Der Befehl löscht das Verzeichnis "Schrott" samt aller Unterverzeichnisse und der darin enthaltenen Daten ohne Rückfragen ("r" steht für "recursive"; "f" steht für "forced").

Aufgaben:

Lösen Sie mit den in dieser Einheit gelernten Kommandos die folgenden Aufgaben:

I. 9.1. Legen Sie innerhalb Ihres Ausgangsverzeichnisses die Unterverzeichnisse "Sicher/Neu/Alles" an.

I. 9.2. Kopieren Sie den *kompletten* Inhalt des Verzeichnisses "Briefe" in das neu angelegte Unterverzeichnis "Sicher/Neu/Alles" und lassen Sie sich alle Kopieraktionen am Bildschirm anzeigen.

I. 9.3. Benennen Sie Ihr Unterverzeichnis "Sicher" in "sik2" um.

I. 9.4. Kopieren Sie alle Dateien Ihres Verzeichnisses "/imbNet/pool/LinuxKurs/A" (bzw. "B" oder "C") – auch die Dateien der Unterverzeichnisse – in das entsprechende Verzeichnis "/imbNet/pool/Eichner/Kurs/LinuxKurs/A" (bzw. "B" oder "C"). Verfolgen Sie den Kopiervorgang am Bildschirm und beachten Sie, dass nur ältere Dateien überschrieben werden dürfen.

I. 9.5. Wechseln Sie in das Verzeichnis "/imbNet/pool/Eichner/Kurs/LinuxKurs/A" (bzw. "B" oder "C") und sehen Sie sich das Resultat an. Wechseln Sie unmittelbar danach in Ihr Ausgangsverzeichnis zurück.

I. 9.6. Vernichten Sie Ihr Arbeitsverzeichnis /imbNet/pool/LinuxKurs/A" (bzw. "B" oder "C").

Lösungen:

I. 9.1. **mkdir -p** *Sicher/Neu/Alles*

I. 9.2. **cp -rv** *Briefe Sicher/Neu/Alles*

I. 9.3. **mv** *Sicher sik2*

I. 9.4. **cp -ruv** $*$ */imbNet/pool/Eichner/Kurs/LinuxKurs/A*

I. 9.5. **cd** */imbNet/pool/Eichner/Kurs/LinuxKurs/A*
ll
cd -
Anmerkung: Falls Sie sich statt meines Lösungsvorschlags das Resultat genauer ansehen wollten und dazu weitere "**cd**"-Befehle innerhalb des Zielverzeichnisses ausführen wollten, sollten Sie mit "**pushd** ." beginnen und statt mit "**cd -**" mit "**popd**" enden.

I. 9.6. **cd** ..
rm -rf *A*

Einheit 5

10 Verzeichnisse spiegeln – "mirrordir"

Der Befehl "**mirrordir**" gehört zwar nicht zu den Standard-Linux-Kommandos, ist aber eine nützliche Ergänzung, die auf dem Computer installiert sein sollte. Wechseln Sie bitte in Ihr Arbeitsverzeichnis "/imbNet/pool/LinuxKurs/A" (bzw. "B" oder "C"), bevor Sie beginnen, die folgenden Beispiele nachzuvollziehen. Bitte ersetzen Sie in den Beispielen "A" durch "B" oder "C", falls Sie an Computer "B" oder "C" arbeiten.

mirrordir . */imbNet/pool/sik/A*	es wird eine komplette Kopie ("Spiegel") des aktuellen Verzeichnisses samt aller Unterverzeichnisse und Daten im Verzeichnis "imbNet/pool/sik/A" angelegt (das Zielverzeichnis muss bereits existieren). **Vorsicht:** Falls Dateien im Zielverzeichnis liegen, werden diese überschrieben. Dateien des Zielverzeichnisses, die nicht im Quellverzeichnis liegen, werden ohne Warnung gelöscht. Es entsteht ein exaktes Spiegelbild des Quellverzeichnisses ("**mirrordir**" steht für "mirror directory").
mirrordir -k */imbNet/pool/sik/A* .	spiegelt die Daten von der Sicherung zurück. Durch die Option "-k" werden Dateien im Zielverzeichnis nicht gelöscht, wenn sie im Quellverzeichnis fehlen. **Anmerkung:** Um dies auszuprobieren, können Sie im Verzeichnis "/imbNet/pool/sik/A" (bzw. "B" oder "C") vorher einige Dateien löschen ("k" steht für "keep files").
mirrordir -v *Verz1 Verz2*	das Verzeichnis "Verz1" wird im Verzeichnis "Verz2" gespiegelt. Alle vom Computer durchgeführten Aktionen werden auf dem Bildschirm angezeigt ("v" steht für "verbose").
mirrordir -v -X *Verz1* . */imbNet/pool/sik/A*	das aktuelle Verzeichnis (".") wird mit Ausnahme des Unterverzeichnisses "Verz1" unter "/imbNet/pool/sik/A" gespiegelt. **Vorsicht:** Nach dem Spiegeln wird es auch im Verzeichnis "/imbNet/pool/sik/A" kein Verzeichnis "Verz1" mehr geben ("X" steht für "exclude path").
mirrordir -R *'[0-9].dat'* . */imbNet/pool/sik/A*	das aktuelle Verzeichnis wird (mit Ausnahme aller Dateien, die nur aus einer Ziffer mit der Eindung '.dat' bestehen) unter "/imbNet/pool/sik/A" gespiegelt. **Vorsicht:** Wie im vorigen Kommando bedeutet dies, dass diese Dateien nach dem Spiegeln im Zielverzeichnis "/imbNet/pool/sik/A" gelöscht werden ("R" steht für "regular expression"; ausführliche Informationen hierzu erhalten Sie in Teil II, Abschnitt 2).

11 Querverweise anlegen und bearbeiten – "ln"

Beim Erstellen von Querverweisen bietet Linux zwei Möglichkeiten: (a) Symbolische Querverweise ("links") sind Zeiger auf Dateien, die nur einmal im Speicher vorhanden sind. (b) Sogenannte harte Querverweise ("hard links"), bei denen die Datei unter mehreren verschiedenen Namen im Speicher liegt. Wildcards (∗?) und Bereichsangaben [ ] können auch beim Jonglieren mit Querverweisen verwendet werden; man beachte jedoch die unter "**cp**" und "**mv**" genannten Einschränkungen. Für Verzeichnisse kann man *nur* symbolische Querverweise verwenden.

ln *dat1 dLink*	erstellt einen Verweis namens "dLink" auf die Datei "dat1" – *per default* handelt es sich dabei um einen "hard link" ("**ln**" steht für "link").
ln -s */imbNet/pool/sik SikLink*	erstellt einen Verweis namens "SikLink", welcher auf das Verzeichnis "/imbNet/pool/sik" zeigt ("s" steht für "symbolic").

Nach Erstellung des Verweises "SikLink" kann man z. B. durch "**cd** SikLink" in das Verzeichnis "/imbNet/pool/sik" wechseln (man hat also eine Querverbindung bzw. eine Abkürzung im Verzeichnisbaum geschaffen); durch nachfolgendes "**cd** .." gelangt man jedoch in das Verzeichnis, welches die Datei "SikLink" enthält, und nicht in das Verzeichnis "/imbNet/pool", wie es man vielleicht erwartet hätte.

Verweise können auch kopiert, umbenannt oder verschoben werden:

cp *dLink Verw*	Legt eine Kopie der Querverweisdatei "dLink" unter dem Namen "Verw" an. Jetzt kann "Verweis" so verwendet werden, wie zuvor "dLink" ("**cp**" steht für "copy").
mv *SikLink sL*	falls "sL" kein bereits existierendes Verzeichnis ist, benennt dieses Kommando "SikLink" in "sL" um, andernfalls erscheint "SikLink" als Unterverzeichnis von "sL" – das Verzeichnis "/imbNet/pool/sik", auf das "SikLink" verweist, wird nicht verändert ("**mv**" steht für "move").
mv *sL AllLinks/*	verschiebt die Querverweisdatei "sL" in das Verzeichnis "AllLinks". **Vorsicht:** ein Querverweis, welcher verschoben wird, funktioniert nur dann noch, wenn er ursprünglich unter Verwendung eines absoluten (und nicht eines relativen) Pfadnamens angelegt wurde.

Das Löschen von Verweisen erfolgt durch:

rm *Verw*	löscht die Datei "Verw", lässt aber das Verzeichnis "/imbNet/pool/sik", auf welches "Verw" zeigt, unangetastet.
rm *SikLink/∗*	**Vorsicht:** dieser Befehl löscht nicht den Verweis "SikLink", sondern den Inhalt des Verzeichnisses "/imbNet/pool/sik", auf das der Verweis zeigt. Analoges gilt für die Befehle "**cp**" und "**mv**", wenn "SikLink/∗" oder etwas ähnliches verwendet wird.

Aufgaben:

Lösen Sie mit den in dieser Einheit gelernten Kommandos die folgenden Aufgaben:

I. 11.1. Legen Sie ein Abbild Ihres Verzeichnisses "/imbNet/pool/LinuxKurs/A" (bzw. "B" oder "C") – einschließlich aller Unterverzeichnisse – unter dem Namen "/imbNet/pool/Eichner/Kurs/LinuxKurs/A" (bzw. "B" oder "C") an. Verhindern Sie dabei bitte das Löschen von Dateien im Zielverzeichnis und verfolgen Sie alle Aktionen an Ihrem Bildschirm.

I. 11.2. Legen Sie einen Querverweis namens "quer" an, der auf Ihr das gespiegelte Unterverzeichnis "/imbNet/sik/A/Verz1" (bzw. "B" oder "C") verweist. Verwenden Sie beim Anlegen des Verweises einen absoluten Pfadnamen.

I. 11.3. Verschieben Sie den Querverweis in Ihr Unterverzeichnis "AllLinks"

I. 11.4. Benutzen Sie den neu erstellten Querverweis "quer", um im Verzeichnis "/imbNet/pool/sik/A/Verz1" (bzw. "B" oder "C") alle Dateien und Unterverzeichnisse zu löschen.

I. 11.5. Benutzen Sie den neu erstellten Querverweis "quer", um alle Dateien (ohne die Unterverzeichnisse) ihres Arbeitsverzeichnisses "/imbNet/pool/LinuxKurs/A" (bzw. "B" oder "C") in das Verzeichnis "/imbNet/pool/sik/A/Verz1" (bzw. "B" oder "C") zu kopieren.

I. 11.6. Legen Sie einen neuen Querverweis "back" im Verzeichnis "/imbNet/pool /sik/A/Verz1" (bzw. "B" oder "C") an, der auf Ihr Arbeitsverzeichnis "/imbNet/pool/LinuxKurs/A" (bzw. "B" oder "C") zeigt.

I. 11.7. Wechseln Sie mit "**cd**" zunächst nach "quer" und dann nach "back" und sehen Sie sich mit "ll" um.

Lösungen:

I. 11.1. **mirrordir -vk** . */imbNet/pool/Eichner/Kurs/LinuxKurs/A*

I. 11.2. **ln -s** */imbNet/pool/sik/A/Verz1 quer*

I. 11.3. **mv** *quer AllLinks*

I. 11.4. **rm -rf** *AllLinks/quer/*∗

I. 11.5. **cp** ∗ *AllLinks/quer/*

I. 11.6. **ln -s** */imbNet/pool/LinuxKurs/A AllLinks/quer/back*

I. 11.7. **cd** *AllLinks/quer*
cd *back*
ll

Einheit 6

12 Eigenschaften von Dateien ermitteln

12.1 Was sind das für Dateien? – "file"

Es liegt wieder mal ein ganzer Haufen Dateien und Sicherungskopien mit den fantasievollsten Namen 'rum und kein Schwein blickt mehr durch, was das alles ist. Höchste Zeit, das Kommando "file" zum Einsatz zu bringen.

file *	ermittelt für jede Datei im aktuellen Verzeichnis den Dateityp. "file" ignoriert die Endung des Dateinamens und untersucht statt dessen den Inhalt der Datei. Dateiklassifikationen sind in der Datei "/usr/lib/magic" gespeichert.
file *V/*.{dat,doc}*	wie voriges Kommando, jedoch eingeschränkt auf Dateien in Verzeichnis "V", welche auf "dat" oder "doc" enden.

12.2 Worte in Textdateien zählen – "wc"

wc *Br.txt*	ermittelt für die Datei Br.txt die Anzahl Zeilen, die Anzahl Worte und die Anzahl Zeichen ("**wc**" steht für "word count").
wc -l *Br.txt*	wie voriges Kommando, jedoch nur Ausgabe der Zeilen ("l" steht für "lines").
wc -w *Br.txt*	wie voriges Kommando, jedoch nur Ausgabe der Worte ("w" steht für "words").
wc -c *Br.txt*	wie voriges Kommando, jedoch nur Ausgabe der Zeichen ("c" steht für "characters").

12.3 Textdateien unter der Lupe – "strings", "ctags" and "ispell"

strings -n *12 p.exe*	durchsucht die Datei "p.exe" (nützlich für nicht-Textdateien) nach Zeichenketten von einer Mindestlänge von 12 und gibt diese auf dem Bildschirm aus ("n" steht für "number of bytes").
ctags -x *pr.c > Z*	dieses Kommando für "C"-Programmierer durchsucht die Datei "pr.c" nach allen Deklarationen von Funktionen, Variablen etc. und gibt sie alphabetisch sortiert mit Zeilennummer in die Datei "Z" aus ("x" steht für "context", d. h. Zeilennummer, Dateiname etc.).
ispell -S *letter.txt*	untersucht die Textdatei "letter.txt" auf Fehler in der englischen Rechtschreibung, macht Veränderungsvorschläge und führt die Verbesserungen auf Wunsch durch ("S" steht für "sort suggestions by likelihood of being correct"). **Anmerkung:** Die wichtigsten interaktiven Kommandos werden zusammen mit dem Text eingeblendet. Weitere Möglichkeiten erhält man, wenn man ein "?" eingibt.
ispell -t -S *file.tex*	nützliche Variante des vorigen Kommandos für TEX- oder LATEX-Benutzer ("t" steht für "expect TEX or LATEX input").

12.4 Unterscheiden sich diese Dateien? – "cmp" and "diff"

Sicherungskopien von Publikations- oder Programmtexten sind doch 'was feines. Man kann immer wieder darauf zurückkommen, nachdem man etwas vermurkst hat – wenn man nur wüsste, was man seit der vorletzten Version alles verändert hat. War da nicht auch eine der dringend nötigen Verbesserungen dabei? Man könnte die 27 Seiten ausdrucken und kritisch durchlesen – oder vielleicht die Seiten übereinanderliegend gegen das Licht halten – oder lieber eines der folgenden Kommandos verwenden?

cmp *A1 A2* vergleicht die Dateien "A1" und "A2" bitweise. No news is good news: Nur wenn Unterschiede da sind, gibt das Programm aus, beim wievielten Zeichen erstmals ein Unterschied auftritt und in welcher Zeile dieses geschieht ("**cmp**" steht für "compare").

diff *A1 A2* vergleicht die Dateien "A1" und "A2" und gibt die Zeilen aus, in denen sich die Dateien unterscheiden ("**diff**" steht für "difference").

Anmerkung: Das Programm "**diff**" gibt zunächst die Zeilennummern (oder -bereiche) von "A1" und "A2" aus, wo sich die Dateien unterscheiden. Danach kommen (blockweise) die unterschiedlichen Zeilen – zunächst die aus der Datei "A1", eingeleitet von einem <-Zeichen, dann die aus der Datei "A2", eingeleitet von einem >-Zeichen. Mit dem Kommando "**diff**" können auch multiple Vergleiche durchgeführt werden:

A1	A2	Ergebnis von "**diff** *A1 A2*"
Datei	Datei	die zwei Dateien werden verglichen
Datei	Verzeichnis	liegt im Verzeichnis eine mit "A1" gleichnamige Datei, so wird diese mit "A1" verglichen
Verzeichnis	Verzeichnis	für jede Datei, die gleichnamig in "A1" und in "A2" vorkommt, wirdein Vergleich durchgeführt

Nachfolgend noch ein paar brauchbare Optionen zu diesem Kommando ("**diff**" hat außerdem Optionen für "C"-Dateien, die von mir aber nicht besprochen werden).

diff -wi *A1 A2* wie "**diff** *A1 A2*", jedoch werden Unterschiede in der Großschreibung sowie in Leerzeichen und -zeilen ignoriert ("i" steht für "ignore case sensitivity", "w" steht für "ignore whitespace").

diff -rs *Verz1 Verz2* vergleicht alle Dateien in "Verz1" (und Unterverzeichnissen) mit gleichnamigen Dateinen in "Verz2" (und Unterverzeichnissen) und gibt nur die Namen und Pfade der Dateien aus, welche sich unterscheiden ("r" steht für "recursive", "s" steht für "summary": es werden nur die Dateinamen mit dem Vermerk "identical" oder "different" ausgegeben).

diff -I *'comment' A1 A2* vergleicht die Dateien "A1" und "A2", ignoriert dabei aber alle Zeilen, welche die Zeichenkette "comment" enthalten ("I" steht für "ignore regular expression"; weitere Informationen zu "Eintragregular expressions" erhalten Sie in Teil II, Abschnitt 2).

12.5 Was haben diese Dateien gemeinsam? – "comm"

Mit dem Befehl "**comm**" können zwei Textdateien (zeilenweise) verglichen werden. Das Programm gibt alle in den beiden Dateien vorkommenden Zeilen in drei Spalten am Bildschirm aus: In der ersten Spalte stehen die Zeilen, welche nur in der ersten Datei vorkommen, in der zweiten stehen die Zeilen, welche nur in der zweiten Datei vorkommen, und in der dritten stehen die gemeinsamen Zeilen. Die Ausgabespalten lassen sich mit den Optionen "-1", "-2" und "-3" (sowie jeder möglichen Kombination davon) abschalten.
Anmerkung: Leider liefert "**comm**" nur dann zufriedenstellende Resultate, wenn die verglichenen Dateien hinreichend ähnlich sind. Ansonsten werden gemeinsame Zeilen nicht entdeckt. Besonders störempfindlich scheint "**comm**" zu sein, wenn in einer Datei große Teile des Anfangs fehlen (es wird also leider doch nicht jede Zeile mit jeder verglichen).

comm *-12 A B > AB*	vergleicht die Dateien "A" und "B" und schreibt die Zeilen, welche in beiden Dateien identisch sind, in die Datei "AB" ("**comm**" steht für "common").
comm *-23 A B > NurA*	vergleicht die Dateien "A" und "B" und schreibt die Zeilen, welche nur in "A" vorkommen, in die Datei "NurA".

Aufgaben:

Lösen Sie mit den bisher gelernten Kommandos die folgenden Aufgaben:

I. 12.1. Das Manuskript "paper" ist fertig. Der Verlag möchte wissen, wieviele Zeichen und Worte ich verwendet habe.

I. 12.2. Der Chef hat mit seinem "Mac" Dateien angelegt und im Verzeichnis "Dringend" abgespeichert – natürlich ohne aussagekräftige Dateiendungen. Welches sind die "JMP"- und welches die "Word"-Dokumente und wo ist meine LaTeX-Datei abgeblieben?

I. 12.3. Offensichtlich ist "MalG148" ein M$-Word-Dokument. Dem Dateinamen nach könnte es die gesuchte Arbeit über Malaria sein. Entfernen Sie die binären Steuerzeichen und werfen einen kurzen Blick auf den in "MalG148" gespeicherten Text, um zu sehen, ob es dort wirklich um Malaria geht.

I. 12.4. Ich habe der Programmdatei "prog.pas" einige Zeilen hinzugefügt und die neue Datei unter "progNeu.pas" abgespeichert. Welche Zeilen waren das?

I. 12.5. Durchsuchen Sie das Verzeichnis "Verz1" samt aller Unterverzeichnisse nach Sicherungskopien namens "paper.txt" und vergleichen Sie die gefundenen Dateien mit der außerhalb liegenden, endgültigen Fassung "paper.txt". Welche Dateien unterscheiden sich von der endgültigen Fassung und welche sind identisch mit ihr? Leerzeichen und -zeilen sowie Groß- und Kleinschreibung sollen bei diesem Vergleich vernachlässigt werden.

I. 12.6. Die harte Nuss:
Ich habe den Datensatz "Daten.txt" noch einmal auf Fehler überprüft und einige Änderungen durchgeführt. Der verbesserte Datensatz ist unter "Daten.neu" abgespeichert. Leider musste ich meine Arbeit unterbrechen und weiß jetzt nicht mehr, wie weit ich gekommen bin. In der wievielten Zeile habe ich die letzte Änderung gemacht? Versuchen Sie eine Lösung ohne "**comm**" zu finden.

Lösungen:

I. 12.1. **wc** *paper* ups, klingt seltsam – aber manchmal hat man den Eindruck

I. 12.2. **file** *Dringend/**

I. 12.3. **strings -n** *20 MalG148*

I. 12.4. **comm** *-12 prog.pas progNeu.pas*

I. 12.5. **diff -wirs** *paper.txt Verz1*

I. 12.6. **tac** *Daten.txt > Alt.inv*
tac *Daten.neu > Neu.inv*
cmp *-12 Alt.inv Neu.inv*
Jetzt weiß ich, in welcher Zeile beim invertierten Datensatz, die *erste* Änderung auftritt und muss nur noch wissen, wie viele Zeilen der Datensatz insgesamt hat. Also noch Zeilen zählen:
wc *Daten.txt*

Natürlich ist das nicht die einzig mögliche Lösung. Man hätte auch mit "**comm** *-12 Daten.txt Daten.neu*" vergleichen können und dann z. B. mit "**less -N -p** *'Zeichenkette' Daten.txt*" nach der letzten geänderten Zeile suchen können. Nur leider habe ich die Erfahrung gemacht, dass "**comm**" manchmal Unterschiede übersieht.

Einheit 7

13 DOS-Disketten mit "mtools" bearbeiten

Die hier vorgestellten "**mtools**" gehören zwar nicht zu den Linux-Standardkommandos, stellen aber eine sehr nützliche Programmsammlung dar: Beim Umgang mit DOS-formatierten Disketten lassen sich die meisten M$-DOS-Kommandos mehr oder weniger problemlos anwenden, wenn man dem Kommandonamen ein "m" vorausstellt. Wildcards (*?) und Bereichsangaben [ ] lassen sich wie gewohnt verwenden.
Vorsicht: Dateinamen unter DOS bestehen aus maximal 8 Zeichen (Basis), eventuell gefolgt von einem Punkt und maximal 3 weiteren Zeichen (Erweiterung). Für Dateien, die von Linux auf die Diskette kopiert werden, wird auf der Diskette die korrekte Schreibweise als Zusatzinformation mit abgespeichert, beim Zugriff mit Wildcards kann es jedoch zu Überraschungen kommen: z. B. wird die überlange Datei "LangerDateiNamen" zwar nicht mit "**mdir** *a:/*n*", wohl aber mit "**mdir** *a:/D**" angezeigt, da der Name von DOS nach 8 Zeichen verstümmelt wird.
Vorsicht: Unter DOS wird Groß- und Kleinschreibung nicht unterschieden. Auf der Diskette können also *nicht* zwei Dateien im gleichen Verzeichnis liegen, deren Namen sich lediglich in der Großschreibung unterscheiden (z. B. "Brief.TXT" und "brief.txt").

13.1 Verzeichnisse einer DOS-Diskette lesen – "mdir"

mdir	listet den Inhalt des obersten Verzeichnisses der Diskette auf ("**mdir**" steht für "MS-DOS directory").
mdir *a:/[0-9]**	listet alle Dateien des obersten Verzeichnisses der Diskette auf, deren Namen mit einer Ziffer beginnen.
mdir *a:/*txt*	**Vorsicht:** wenn man nach einem besonderen Dateiende (hier "txt") sucht, bekommt man nur die Dateien angezeigt, deren Namen *nicht* beim Umwandeln in DOS-Dateien verstümmelt wurden.
mdir *a:/br/*	listet den Inhalt des Verzeichnisses "br" der Diskette auf.

13.2 Dateien auf DOS-Disketten kopieren – "mcopy"

mcopy * *a:*	kopiert alle Daten des aktuellen Verzeichnisses auf die Diskette. Liegen im aktuellen Verzeichnis Unterverzeichnisse, so werden auch diese mit den darin befindlichen Daten kopiert; der Inhalt deren Unterverzeichnisse wird jedoch nicht kopiert ("**mcopy**" steht für "MS-DOS copy").
mcopy -v *a:* .	kopiert alle Dateien des aktuellen Verzeichnisses der Diskette in das aktuelle Verzeichnis der Festplatte. Unterverzeichnisse werden nicht kopiert. Sind auf der Diskette Zusatzinformationen zu langen Dateinamen gespeichert, so werden die Dateinamen restauriert. Kopieraktionen werden am Bildschirm angezeigt ("v" steht für "verbose").
mcopy -v *a:/x a:/xx*	kopiert die Datei "x" von der Diskette als Datei "xx" auf die Diskette.

mcopy *[a-z]* a:/brief* — kopiert alle Dateien, die mit einem Kleinbuchstaben beginnen in das Verzeichnis "brief" auf der Diskette. Der Computer fragt nach, bevor bestehende Dateien überschrieben werden (wichtig bei Groß/Klein-Schreibungs-Konflikten).
Anmerkung: Der Bereich "a-z" enthält auch die Umlaute "ä", "ö", "ü" und das deutsche "ß".

13.3 Dateien auf DOS-Disketten umbenennen oder löschen – "mren" and "mdel"

mren -v *a:/xx a:/yy* — benennt die Datei "xx" auf der Diskette in "yy" um und zeigt den Vorgang am Bildschirm an ("**mren**" steht für "MS-DOS rename"; "v" steht für "verbose").

mdel -v *a:/old/dat** — löscht alle Dateien im Verzeichnis "old" der Diskette, welche mit "dat" beginnen, und zeigt die Löschaktionen am Bildschirm an ("**mdel**" steht für "MS-DOS delete"; "v" steht für "verbose").

13.4 Verzeichnisse auf DOS-Disketten anlegen oder löschen – "mmd", "mrd" and "mdeltree"

mmd *a:/brief* — legt das Verzeichnis "brief" auf der Diskette an ("**mmd**" steht für "make MS-DOS directory").
Anmerkung: Wildcards funktionieren hier nicht; z. B. "**mmd** *a:/V[1-3]*" führt nicht zum erwünschten Ergebnis.

mrd -v *a:/old* — löscht das Verzeichnis "old" auf der Diskette falls dieses leer ist und gibt den Vorgang am Bildschirm aus ("**mrd**" steht für "MS-DOS remove directory"; "v" steht für "verbose").

mdeltree *a:/[x-z]* — löscht die Verzeichnisse "x", "y" und "z" auf der Diskette samt ihrer Unterverzeichnisse ohne nachzufragen.
BUG: Das sollte eigentlich funktionieren; durch einen bug wird das letzte Verzeichnis "z" allerdings nicht gelöscht ("**mdeltree**" steht für "MS-DOS delete directory tree").

mdeltree -v *a:* — **Vorsicht:** Dies ist die brutale Variante, die komplette Diskette samt aller Verzeichnisse und Unterverzeichnisse zu löschen, ohne einmal nachgefragt zu werden. Der Befehl zeigt alle Löschaktionen am Bildschirm an ("v" steht für "verbose").

13.5 Auf ein anderes Verzeichnis der DOS-Diskette wechseln – "mcd"

Wenn einem die Pfadeingaben (z. B. "a:/br/old/M/Meinzel/") zu mühsam werden, kann man auch das aktuelle Verzeichnis auf der Diskette ändern. Alle vorne beschriebenen Befehle gelten dann (soweit keine absoluten Pfadangaben gemacht wurden) für das jeweils aktuelle Verzeichnis auf der Diskette.

mcd *a:/br/old/M* — setzt das aktuelle Verzeichnis der Diskette auf "a:/br/old/M" ("**mcd**" steht für "MS-DOS change directory").

mcd ..	wechselt das aktuelle Verzeichnis der Diskette auf das höher-gelegene Verzeichnis.

13.6 DOS-Disketten formatieren – "mformat"

mformat *a:*	formatiert eine Diskette im DOS-Format. **Vorsicht:** Alle bestehenden Dateien und Verzeichnisse werden ohne Nachfragen gelöscht ("**mformat**" steht für "MS-DOS format").

14 Dateikonversion DOS ⇄ Linux

Textdateien, die unter DOS erstellt wurden, müssen geringfügig verändert werden, damit sie sinnvoll unter Linux verwendet werden können. Soweit diese Konversion nicht von den "**mtools**" automatisch durchgeführt wurde, müssen die Dateien nachbearbeitet werden:

dos2unix *dat1.txt*	konvertiert die DOS-Datei "dat1.txt" in das von Unix benötigte Format.
unix2dos *dat1.txt*	konvertiert die Unix-Datei "dat1.txt" in das von DOS benötigte Format.

Aufgaben:

Lösen Sie mit den in dieser Einheit gelernten Kommandos die folgenden Aufgaben:

I. 14.1. Kopieren Sie alle Dateien Ihres Festplattenverzeichnisses "Verz" und der darin enthaltenen Unterverzeichnisse auf die Diskette. Lassen Sie sich die Kopieraktionen auf dem Bildschirm anzeigen.

I. 14.2. Lassen Sie sich den Inhalt der Diskette (ohne Unterverzeichnisse) anzeigen.

I. 14.3. Lassen Sie sich alle Dateien des Verzeichnisses "daten" auf der Diskette anzeigen, welche auf ".jmp" enden.

I. 14.4. Benennen Sie die Datei "murks" auf der Diskette in "schrott" um.

I. 14.5. Löschen Sie auf der Diskette alle Dateien im Verzeichnis "daten", welche mit einem "a", "e" oder "u" beginnen. Lassen Sie sich die Löschaktionen am Bildschirm anzeigen.

I. 14.6. Löschen Sie alle Daten auf der Diskette. Lassen Sie sich alle Löschvorgänge am Bildschirm anzeigen.

I. 14.7. Formatieren Sie die Diskette neu.

Lösungen:

I. 14.1. **mcopy -v** *Verz/* a:*

I. 14.2. **mdir** *a:*

I. 14.3. **mdir** *a:/daten/*.jmp*
leider klappt dieses Kommando nicht, wenn überlange Dateien auf ".jmp" enden

I. 14.4. **mren** *a:/murks a:/schrott*

I. 14.5. **mrm -v** *a:/daten/[aeu]**

I. 14.6. **mdeltree -v** *a:*

I. 14.7. **mformat** *a:*

Einheit 8

Teil II

Suchen und Finden

1 Texte durchsuchen – "grep", "fgrep" and "egrep"

Zum Auffinden von Stichworten in Textdateien haben Sie bereits das Kommando "**less -N -p** " kennengelernt. Ein weiteres äußerst nützliches Kommando hierfür ist "**grep**" sowie seine Varianten "**fgrep**" und "**egrep**". "**egrep**" durchsucht eine Datei zeilenweise nach einem Suchbegriff und gibt *per default* die Zeile, welche den Suchbegriff enthält, auf dem Bildschirm aus. Beim Suchbegriff handelt es sich um eine "regular expression" (näheres siehe unten sowie nächste Einheit). Alle Befehle "**grep**", "**egrep**" und "**fgrep**" haben die gleichen Optionen und in etwa das gleiche Verhalten. Da "**egrep**" in allen wichtigen Fällen ein größeres Repertoire an "regular expressions" abdeckt als "**grep**", werden wir hier dieses Kommando verwenden (außerdem soll "**egrep**" schneller sein als "**grep**"). Die Anführungszeichen um den Suchbegriff sind nicht immer zwingend nötig, sind aber empfehlenswert, um Interpretationskonflikte zu vermeiden, wenn Leerzeichen oder andere Sonderzeichen im Suchbegriff vorkommen.

egrep *'hallo' D*	sucht in der Datei "D" nach "hallo" und gibt alle Zeilen, in denen "hallo" vorkommt, auf dem Bildschirm aus.
egrep -n *'hallo' D*	wie das vorige Kommando, jedoch wird bei der Ausgabe der Zeilen zuerst die Zeilennummer angegeben ("n" steht für "line number").
egrep -i *'hallo' D*	sucht in der Datei "D" nach "hallo", wobei Groß- und Kleinschreibung ignoriert werden ("i" steht für "ignore case").
egrep -w *'in' D*	durch Angabe der Option "w" wird der Suchbegriff "in" nur einzeln gesucht und nicht als Teil eines größeren Wortes wie z. B. "ein" ("w" steht für "word") **Anmerkung:** "Wörter" bestehen aus Buchstaben, Ziffern und dem Unterstrich "_".).
egrep -x *'Alles' D*	durch Angabe der Option "x" wird der Suchbegriff "Alles" in der Datei "D" nur dann gefunden, wenn er der einzige Eintrag in einer Zeile ist.

Wenn man gleich ganze Verzeichnisse durchsuchen möchte oder nur wissen möchte, ob ein Suchbegriff überhaupt in einer Datei vorkommt, empfehlen sich die folgenden Optionen:

egrep -r *'hallo' ∗*	sucht in allen Dateien (auch in den Unterverzeichnissen) nach dem Suchbegriff "hallo" ("r" steht für "recursive").
egrep -rl *'hallo' ∗*	wie das vorige Kommando, jedoch werden nur die Namen der Dateien ausgegeben, in denen der Suchbegriff gefunden wurde ("l" steht für "list file names").
egrep -c *'hallo' D*	sucht in der Datei "D" nach "hallo" und gibt auf dem Bildschirm aus, wie oft der Suchbegriff in der Datei gefunden wurde ("c" steht für "count").

2 Kurze Einführung in "regular expressions"

2.1 Bereichsangaben "[ ]"

Bereichsangaben [ ] ("character classes") haben Sie bereits bei den Kommandos "**ls**", "**less**", "**cp**", "**mv**" und "**rm**" kennengelernt – oft in Kombination mit den Wildcardzeichen "∗" und "?" (Wildcards gehören *nicht* zu den "regular expressions"). Eine Bereichsangabe kann zwar jede beliebige Ansammlung von Buchstaben, Zahlen und sonstigen Zeichen haben, sie steht aber immer für ein *einzelnes* Zeichen:

egrep -n *'[hier]' D*	sucht in der Datei "D" alle Buchstaben "h", "i", "e" und "r" (also nicht nur das Wort "hier") und gibt die Zeilen mit Treffern samt Zeilennummern auf dem Bildschirm aus.
egrep -i *'gr[ea]y' D*	sucht in der Datei "D" alle Worte "grey" und "gray" sowie alle Varianten, die durch Verwendung von Großbuchstaben entstehen.
egrep -w *'[0-9][0-9]' D*	sucht in der Datei "D" alle (exakt) zweistelligen Zahlen.
egrep *'[-_]' D*	**Vorsicht:** Wenn vor einem "-" kein Zeichen steht, kann "-" auch nicht als "von ... bis ..." interpretiert werden. Dieser Befehl sucht nach allen Vorkommnissen von "-" und "_" in der Datei "D".

Möchte man den Inhalt einer Bereichsangabe negieren, so beginnt man sie mit dem Zeichen "^":

egrep -c *'[^a-z A-Z 0-9]' D*	sucht in der Datei "D" alle Sonderzeichen (keine Buchstaben, Ziffern oder Leerzeichen) sind und gibt die Anzahl der Zeilen, die solche Zeichen enthalten, am Bildschirm aus.
egrep -i *'q[^u]' ∗.{txt,tex}*	sucht in allen Dateien, welche auf ".txt" oder ".dat" enden, die Vorkommnisse von klein oder groß geschriebenem "q", denen kein "u" folgt. **Ausnahme:** "q" am Ende der Zeile wird nicht gefunden, da ihm ja auch kein "Zeichen ≠ u" folgt.
egrep *'[~ ^]' D*	**Vorsicht:** Dieser Befehl sucht in der Datei "D" alle Vorkommnisse von "~" und "^" – das Zeichen "^" wirkt nur dann als Negation, wenn es unmittelbar auf die öffnende Klammer der Bereichsangabe folgt.

2.2 Zeichen abschalten mit "?"

Wenn ein "?" in einer "regular expression" vorkommt, so bedeutet dies, dass das vorausgehende Zeichen an- oder abwesend sein darf:

egrep -i *'colou?r' D*	sucht in der Datei "D" alle "color" und "colour" ungeachtet der Großschreibung.

2.3 Größere Ausdrücke wie ein Zeichen behandeln "( )"

Durch die Verwendung von Klammern "( )" können ganze Ausdrücke so wie ein Zeichen behandelt werden (funktioniert nur mit "**egrep**", nicht mit "**grep**").

egrep *'Ur(groß)?enkel' D*	sucht in der Datei "D" nach den Begriffen "Urgroßenkel" und "Urenkel". Durch die Klammerung wird "groß" wie ein einziges Zeichen behandelt, das durch "?" abgeschaltet werden kann.
egrep *'was ist(los)?[?]' D*	sucht in der Datei "D" nach "was ist los?" und nach "was ist?". Da nach dem zweiten Fragezeichen wörtlich gesucht werden soll, muss es in einer Bereichsangabe versteckt werden.

2.4 Stellen, an denen beliebige Zeichen vorkommen dürfen "."

Anstelle des Wildcardzeichens "?" verwendet man bei "regular expressions" den Punkt ".", um ein *beliebiges* Zeichen darzustellen:

egrep *'1.0' D*	sucht in der Datei "D" alle "1 gefolgt von irgendeinem Zeichen gefolgt von 0", z. B. "1,0", "170", "1A0" oder "1 0".
egrep *'1[.]0' D*	**Vorsicht:** Der Punkt ist nur *außerhalb* von Bereichsangaben ein Sonderzeichen. Das angegebene Kommando sucht in der Datei "D" wörtlich nach "1.0".

Aufgaben:

Benutzen Sie zum Lösen aller Aufgaben den Befehl "**egrep**":

II. 2.1. Wie schreibt man das doch gleich? "sep**ara**te" oder "sep**ere**te"? Oder irgendeine Kreuzung aus den beiden? Suchen Sie alle Vorkommnisse der vier Möglichkeiten in der Datei "D"

II. 2.2. In welchen Dateien (inklusive der Dateien in den Unterverzeichnissen) findet sich der Begriff "Vorsicht"?

II. 2.3. Suchen Sie in den Dateien, deren Namen "DM" oder "Preis" enthalten, alle Preisangaben unter 10 DM, die auf ",99 DM" enden.

II. 2.4. Suchen Sie in den Dateien, deren Namen "DM" oder "Preis" enthalten, alle Preisangaben unter 100 DM, die auf ",99 DM" enden.

II. 2.5. Prozedurdeklarationen in der Programmiersprache "Pascal" enthalten immer das Schlüsselwort "procedure"; Groß- und Kleinschreibung spielen dabei keine Rolle. Geben Sie eine Liste aus, welche alle Prozedurdeklarationen der Datei "ring.pas" sowie die entsprechenden Zeilennummern enthält.

II. 2.6. Datumsangaben könnten in der Form 24-11-98 oder 24.11.98 oder 24/11/98 voliegen. Statt "98" könnte da aber auch "1998" stehen. Suchen Sie nach allen Möglichkeiten für Silvester '99 in der Datei "D".

Lösungen:

II. 2.1. **egrep** *'sep[ae]r[ae]te' D*

II. 2.2. **egrep -rl** *'Vorsicht' **

II. 2.3. **egrep -w** *'[0-9],99 DM' *{DM,Preis}**

II. 2.4. **egrep -w** *'([1-9])?[0-9],99 DM' *{DM,Preis}**

II. 2.5. **egrep -win** *'procedure' ring.pas*

II. 2.6. **egrep** *'31[-./]12[-./](19)?99' D*

(OK zugegeben, das lässt auch Mischformen zu – aber anders geht es mit den Möglichkeiten nicht, die wir in dieser Einheit kennengelernt haben)

Einheit 9

2.5 Zeichen, die mindestens einmal vorkommen müssen "+"

Das Sonderzeichen "+" wird verwendet, um zu zeigen, dass das vorausgehende Zeichen mindestens einmal (oder öfters) vorkommt:

egrep -w *'[1-9]+0' D*	sucht in der Datei "D" alle mindestens zweistelligen Zahlen, die auf "0" enden (*nicht* aber Ziffernfolgen, die Teil eines Namens sind wie z. B.: "das 20te Jahrhundert").
egrep -i *'(ur)+enkel' D*	sucht in der Datei "D" alle Vorkommnisse von "Urenkel", "Ururenkel", "Urururenkel" ... (bei beliebiger Groß- oder Kleinschreibung).
egrep *'1[+]0' D*	**Vorsicht:** Das Pluszeichen ist nur *außerhalb* von Bereichsangaben ein Sonderzeichen. Das angegebene Kommando sucht in der Datei "D" wörtlich nach "1+0". **Anmerkung:** Diesen Befehl kann man auch so schreiben: **egrep** *'1\+0' D*

2.6 Zeichen, die beliebig oft vorkommen dürfen "*"

Das Sonderzeichen "*" wird verwendet, um zu zeigen, dass das vorausgehende Zeichen beliebig oft vorkommen kann. "Beliebig oft" kann dabei auch heißen "gar nicht".

egrep -w *'[1-9][0-9]*' D*	sucht in der Datei "D" alle ganzen Zahlen (nicht aber Ziffernfolgen, die Teil eines Namens sind wie z. B.: "das 18te Jh.").
egrep -wi *'(ur)*enkel' D*	sucht in der Datei "D" alle Vorkommnisse von "Enkel", "Urenkel", "Ururenkel" ... (bei beliebiger Groß- oder Kleinschreibung).
egrep *'1[*]0' D*	**Vorsicht:** Der Stern ist nur *außerhalb* von Bereichsangaben ein Sonderzeichen. Das angegebene Kommando sucht in der Datei "D" wörtlich nach "1*0". **Anmerkung:** Diesen Befehl kann man auch so schreiben: **egrep** *'1*0' D*

2.7 Zeichen, die mit einer bestimmten Häufigkeit vorkommen müssen "{ }"

Wie sich zum Beispiel auch im Abschnitt 2.9 zeigen wird, ist es oft nützlich, genau angeben zu können, wie oft ein bestimmtes Zeichen (oder eine bestimmte Zeichenkette) vorkommt. Dazu muss man unmittelbar hinter das Zeichen (oder den von Klammern umgebenen Ausdruck) in geschweiften Klammern angeben, wie oft das vorausgegangene Zeichen mindestens und höchstens vorkommen soll. Zum Beispiel sucht 's{3}' nach 'sss'. Die Angabe 's{2,}' sucht nach *mindestens* zwei aufeinanderfolgenden "s" und 's{,4}' sucht nach *höchstens* vier aufeinanderfolgenden "s". Entsprechend sucht 's{2,4}' nach mindestens zwei und höchstens vier aufeinanderfolgenden "s".

egrep -n *'[1-9]0{6},' D*	sucht in der Datei "D" nach Zahlen, die vor dem Komma exakt sechs Nullen haben und geben die entsprechende Zeile samt Zeilennummer aus.
egrep -wi *'(ur){2,3}enkel' D*	sucht in der Datei "D" alle Vorkommnisse von "Ururenkel" und "Urururenkel" (bei beliebiger Groß- oder Kleinschreibung).
egrep -n *'[0-9]{8,}' D*	sucht in der Datei "D" alle Zahlen mit mindestens 8 Stellen. **Anmerkung:** Diese Angabe veranlasst "**egrep**" nach mindestens acht aufeinanderfolgenden Ziffern zu suchen, also nicht notwendigerweise nach acht identischen Ziffern.
egrep -n *'[,.;:!?]{2}' D*	sucht in der Datei "D" nach Zeilen, in denen zwei Satzzeichen aufeinanderfolgen.

2.8 Worte erkennen "\< \>"

Ungeachtet der Möglichkeit, "**egrep -w** " zu benutzen, gibt es bei "regular expressions" auch eine eingebaute Worterkennung. Solange man nur nach einem einzelnen Wort sucht, kann man natürlich auch "**egrep -w** " benutzen. Spätestens dann, wenn man kompliziertere Ausdrücke mit mehreren Worten sucht, erweisen sich jedoch die Wortklammern als überlegen. Dabei wird jedes Zeichen, das kein Buchstabe, keine Zahl und kein Unterstrich "_" ist als Trennzeichen zwischen Worten interpretiert. Bei "regular expressions" wird die von "\<" und "\>" eingeschlossene Zeichenkette als "Wort" interpretiert.

egrep *'\<ist\>' D*	sucht in der Datei "D" nach dem *Wort* "ist" (Ausdrücke wie "Das ist's!" und "Es ist!" werden erkannt, während "Mist" oder "L_ist_01" nicht erkannt werden).
egrep -in *'\<[a-z"]{25,}\>' D*	Mammutjagd: Dieser Befehl sucht in der Datei "D" nach Worten, welche aus mindestens 25 Zeichen bestehen und gibt die gefundenen Zeilen und deren Zeilennummern aus. **Anmerkung:** Die Anführungszeichen habe ich deshalb in die Bereichsangabe mit aufgenommen, weil diese in LaTeX ein Bestandteil von Worten sind.
egrep -n *"\<so'n'Murks\>" D*	sucht nach dem Ausdruck "so'n'Murks". **Anmerkung:** Es wird nicht geprüft, ob die angegebene Zeichenkette nur für "Worte" vorgesehene Zeichen enthält, sondern nur darauf, dass sie nach außen hin durch Zeichen abgegrenzt ist, die nicht zu einem "Wort" gehören dürfen ("ssso'n'Murks" wird also *nicht* erkannt). **Anmerkung:** Wir müssen hier doppelte Anführungszeichen verwenden, da im Suchbegriff die einfachen vorkommen.
egrep -in *'\<die\>[^,]+\<welche\>' D*	Fehlersuche: Gibt es eine Zeile in der Datei "D", in welcher "die" und "welche" nicht von einem Komma getrennt sind?

Aufgaben:

Benutzen Sie zum Lösen aller Aufgaben den Befehl "**egrep**": Verwenden Sie dabei bitte nicht die "**-w**"-Option von "**egrep**" sondern – soweit nötig – die "regular expressions"-Klammern.

II. 2.1. Was ist die Bedeutung der folgenden "regular expressions"?
[ur]enkel
(ur)enkel
(ur)?enkel
(ur)∗enkel
(ur)+enkel

II. 2.2. In *wie vielen Zeilen* der Datei "D" stehen nur Punkte '.' und/oder Striche '-'?

II. 2.3. Finden Sie in der Datei "D" alle Zeilen mit sechs- bis acht-stelligen Zahlen.

II. 2.4. Finden Sie in der Datei "D" alle Zeilen, in denen zwei aufeinanderfolgende Zahlen durch mindestens ein Leerzeichen getrennt sind.

II. 2.5. Finden Sie die Zeilen in der Datei "D", in welchen zunächst das Wort "Ziffer" und dahinter irgendwo das Wort "Klammer" vorkommt.

II. 2.6. Finden Sie die Zeilen in der Datei "D", in welchen zweimal das Wort "dies" vorkommt, wobei zwischen dem ersten und dem zweiten Auftreten mindestens 20 und höchstens 30 Zeichen liegen sollen (die Groß- und Kleinschreibung soll dabei vernachlässigt werden).

Lösungen:

II. 2.1. [ur]enkel = uenkel, renkel
(ur)enkel = urenkel
(ur)?enkel = enkel, urenkel
(ur)∗enkel = enkel, urenkel, ururenkel ...
(ur)+enkel = urenkel, ururenkel, urururenkel ...

II. 2.2. **egrep -cx** *'[.-]+' D*

II. 2.3. **egrep** *'\<[1-9][0-9]{5,7}\>' D*

II. 2.4. **egrep** *'[0-9] +[0-9]' D*

II. 2.5. **egrep** *'\<Ziffer\>.∗\<Klammer\>' D*

II. 2.6. **egrep -i** *'\<dies\>.{20,30}\<dies\>' D*

2.9 Rückbezüge in "regular expressions"

Einheit 10

Eine weitere Möglichkeit von "regular expressions" ist die Verwendung der Rückbezugsoperatoren \1, \2, ... Einem "\1" muss mindestens ein Paar Klammern "( )" vorausgegangen sein. In der "regular expression" '(ur)\1' wird zunächst nach der Zeichenkette "ur" gesucht. Die Zeichenkette "ur" wird durch die Klammern zu einem Ausdruck zusammengefasst und genau dieser in Klammern stehende Ausdruck muss wegen "\1" noch einmal gefunden werden. '(ur)\1' ist also identisch mit 'urur' und '(ur)\1\1' ist identisch mit 'ururur'. Um es ganz genau zu sagen: "\1" steht für den Text, der beim Auflösen des ersten Klammerpaares gefunden wurde, "\2" steht für den Text des zweiten Klammerpaares, usw.

egrep *'([a-z])\1{2}' D*	Rechtschreibreform? Dieser Befehl sucht in der Datei "D" nach Kleinbuchstaben, die drei mal hintereinander vorkommen.
egrep -w *'([0-9])\1{2,}' D*	Schnapszahlen: Dieser Befehl sucht in der Datei "D" nach mindestens dreistelligen Zahlen, in denen nur eine Ziffer vorkommt (z. B. 999).
egrep -n *'([,.;:!?])\1' D*	sucht in der Datei "D" nach Satzzeichen, die mindestens doppelt eingegeben wurden, und gibt die gefundenen Zeilen mit Zeilennummern aus.
egrep -n *'([0-9])([0-9])([0-9])\1\2\3' D*	Wiederholungen: Dieser Befehl sucht in der Datei "D" nach Zahlen, bei denen drei aufeinanderfolgende Ziffern zwei mal vorkommen, wie z. B. in "123123". **Anmerkung:** Das wäre auch leichter gegangen: **egrep -n** *'([0-9]{3})\1' D*
egrep -n *'([0-9])([0-9])([0-9])\3\2\1' D*	wie der vorige Befehl, jedoch soll die zweite Dreiergruppe spiegelverkehrt zur ersten sein, wie z. B. in "123321".

Vorsicht: Der durch "\1" angegeben Begriff muss auch bezüglich der Groß- und Kleinschreibung mit dem vorausgegangenen Klammerausdruck identisch sein. Da die "regular expression" zuerst abgearbeitet wird und dann erst das Resultat an "**egrep**" zurückgeliefert wird, kann auch durch Verwendung der "**-i**"-Option nicht mehr erreicht werden, dass "\1" bezüglich der Großschreibung vom vorausgegangenen Klammerausdruck abweicht.

2.10 Heute schon gestottert?

Es gilt als schlechter Stil und klingt holprig, wenn man das gleiche Wort zwei mal unmittelbar unmittelbar hintereinander werwendet (außerdem ist dies ein typischer "copy and paste"-Fehler). Die, die die Sprache so gebrauchen, muss man ja nicht unbedingt nachahmen.

egrep -n *'(\<[A-Za-z]+\>) \<\1\>' D*	sucht in der Datei "D" nach Zeilen, in denen – durch ein Leerzeichen getrennt – zweimal das gleiche Wort hintereinander vorkommt. **Anmerkung:** Die Wortklammern um "\1" sind nötig, da andernfalls auch Texte wie "die Diebe" gefunden werden.
egrep -n *'(\<[A-Za-z]+\>)[,]+\<\1\>' D*	wie das vorige Kommando, jedoch können beliebig viele Leerzeichen (und Kommata) zwischen den Worten stehen.

Eine weitere Stottermöglichkeit besteht darin, mit dem neuen Wort so zu beginnen, wie man mit dem alten geendet hat – allem guten Rat ungeachtet achtet man eben doch nicht immer auf solche Feinheiten.

egrep -n *'([a-z]{5,}) +\1' D*	sucht in der Datei "D" nach Worten, die ebenso beginnen, wie das vorige Wort endete. Der gemeinsame Bereich muss mindestens 5 Buchstaben lang sein. Zwischen den Worten soll mindestens ein Leerzeichen sein.

Natürlich sollte man es (außer bei den kleinen Wörtchen) auch vermeiden, das gleiche Wort in einem Satz mehrmals zu wiederholen. Da ich die Anwohnheit habe, pro Satz eine Zeile zu verwenden, kann man dies mit dem folgenden Befehl überprüfen:

egrep -n *([A-Za-z]{15,}).*\1' D*	Sucht in der Datei "D" nach Zeilen, in welchen Zeichenketten mit mindestens 15 Buchstaben mindestens zwei mal identisch vorkommen. **Anmerkung:** Auf die Wortklammern habe ich verzichtet, um auch Ausdrücke mit unterschiedlichen Vor- oder Nachsilben zu finden.

2.11 Anfang "^" und Ende "$"

"regular expressions" erlauben ebenfalls, bestimmte Strukturen am Anfang oder Ende der Zeile zu suchen: Das Hütchen "^" ("caret") am Anfang einer "regular expression" sorgt dafür, dass nur am Zeilenanfang nach dem nachfolgenden Ausdruck gesucht wird.

egrep -i *'^vor' D*	sucht die Zeilenanfänge der Datei "D" nach dem Begriff "vor" ab.
egrep *'^[a-z]' D*	sucht alle Zeilenanfänge der Datei "D", die mit einem Kleinbuchstaben beginnen.
egrep -i *'^[^a-z]' D*	sucht alle Zeilenanfänge der Datei "D", die nicht mit einem Buchstaben beginnen. **Vorsicht:** Das caret "^" innerhalb der Bereichsangabe hat eine völlig andere Bedeutung als das vor der Bereichsangabe.
egrep *'^[^^]' D*	**Horrorkabinett:** Dieser Befehl sucht in der Datei "D" alle Zeilenanfänge ab (erstes "^"). Es sucht nach einem Zeichen, das durch die Bereichsangabe definiert ist. Dieses Zeichen ist *nicht* (wegen des ersten "^" in der Bereichsangabe) das Zeichen "^" (das ist das zweite "^" in der Bereichsangabe). Alles klar?

Das Dollarzeichen "$" am Ende einer "regular expression" sorgt dafür, dass nur am Zeilenende nach dem vorausgegangenen Ausdruck gesucht wird.

egrep *'am Ende$' D*	sucht die Zeilenenden der Datei "D" nach dem Begriff "am Ende" ab.
egrep *',$' D*	sucht nach Zeilenenden der Datei "D", die aus einem Komma "," bestehen.

Wenn eine "regular expression" mit dem Zeilenanfangszeichen "^" beginnt und mit dem Dollarzeichen endet, muss die ganze Zeile mit der "regular expression" übereinstimmen. Man kann für diesen Zweck auch "**egrep -x** " verwenden. Da andere Kommandos diese Option in der Regel nicht anbieten, möchte ich auch hierzu einige Beispiele besprechen:

egrep *'^Kurze Zeile$' D*	sucht in der Datei "D" nach einer Zeile, die nur aus dem Eintrag "Kurze Zeile" besteht. **Anmerkung:** Dieser Befehl ist identisch mit: **egrep -x** *'Kurze Zeile' D*
egrep -i *'^jetzt .* los$' D*	sucht in der Datei "D" nach einer Zeile, die mit "jetzt " beginnt und mit " los" endet.
egrep -i *'^[a-z]+$' D*	sucht in der Datei "D" nach einer Zeile, in der nur Buchstaben und/oder Leerzeichen vorkommen.
egrep *'^ +$' D*	sucht in der Datei "D" nach einer Zeile, die nur Leerzeichen enthält.
egrep *'^$' D*	sucht nach einer völlig leeren Zeile.

2.12 Das eine oder das andere " | "

"Regular expressions" erlauben auch, mehrere Optionen anzugeben, welcher Suchbegriff gefunden werden soll. Um die verschiedenen Alternativen voneinander zu trennen, verwendet man " | ". Anbei einige Beispiele, die helfen sollen, die Gültigkeitsbereiche der verschiedenen Alternativ-Ausdrücke gegeneinander abzugrenzen:

'A1 | A2'	der erste Ausdruck "A1" beginnt am Anfang der "regular expression" und endet am " | "-Zeichen, der zweite "A2" beginnt am " | "-Zeichen und geht bis zum Ende der "regular expression"
'A1 | A2 | A3 | A4'	bietet vier Alternativen: "A1" oder "A2" oder "A3" oder "A4"
'A1(A2 | A3)A4'	zwischen "A1" und "A4" muss entweder "A2" oder "A3" stehen
'A1 | A2(A3 | A4)'	Alternativen können auch geschachtelt sein: diese "regular expression" sucht entweder nach dem Ausdruck "A1" oder nach dem Ausdruck "A2", wobei letzterem aber Ausdruck "A3" oder "A4" folgen muss

Im Zweifelsfall gilt: lieber ein paar Klammerpaare "( )" mehr als weniger. Man beachte jedoch, dass bei Verwendung der Rückbezugsoperatoren "\1", "\2", ... abgezählt werden muss, bei der wievielten sich öffnenden Klammern der Ausdruck beginnt, auf den Bezug genommen werden soll.

egrep -wi *'Murks| Schrott' D*	sucht in der Datei "D" nach Zeilen, die entweder das Wort "Murks" oder das Wort "Schrott" enthalten.
egrep *'\<DM\>|\<[$]\>' D*	sucht in der Datei "D" nach Zeilen, die entweder das Wort "DM" oder das Wort "$" enthalten.
egrep *'^(Anfang| Ende)$' D*	sucht in der Datei "D" nach Zeilen, die nur aus dem Wort "Anfang" oder aus dem Wort "Ende" bestehen.
egrep *'^Anfang| Ende$' D*	**Vorsicht:** dieser Befehl sucht in der Datei "D" nach Zeilen, die mit "Anfang" beginnen und/oder mit "Ende" enden.

2.13 Trouble shooting

"regular expressions" haben im Großen und Ganzen zwar immer die gleiche Struktur, ihre Benutzung (und leider auch ihre Interpretation) unterscheidet sich aber geringfügig von Programm zu Programm. Manche Programme haben einfach nicht den "vollen" Funktionsumfang der "regular expressions" – so kennt "**grep**" beispielsweise keine Klammern "()". Manche Programme haben auch einen etwas befremdlichen Umgang mit den Sonderzeichen der "regular expressions": Oft genügt es, vor die "regular expression" Sonderzeichen jeweils einen backslash "\" zu setzen (man spricht dann von "escaped characters") – damit man den wörtlich gemeinten backslash noch erkennen kann, wird dieser dann oft durch einen doppelten "\" ersetzt, was dem ganzen Ausdruck ein recht bizarres Aussehen verleiht.

2.14 Zusammenfassung

Am Ende der Beschreibung der "regular expressions" ist es jetzt wohl dringend nötig, eine Tabelle zu haben, welche die verschiedenen Sonderzeichen und deren Bedeutung für "**egrep**" zusammenfasst. Zunächst einmal die zwei Zeichen, die innerhalb von Bereichsangaben "[ ]" als Sonderzeichen auftreten können:

Zeichen	Bedeutung als Sonderzeichen	wörtlich
^	Verneinung	nicht am Anfang der Bereichsangabe oder außerhalb als \^
-	von ... bis	am Anfang oder Ende der Bereichsangabe

Außerhalb von Bereichsangaben bibt es darüber hinaus noch eine ganze Reihe von Zeichen mit Sonderbezeichnung. Um nach solchen Zeichen wörtlich zu suchen, genügt es meist, sie entweder in einer Bereichsangabe zu "verstecken" oder einen backslash "\" vorauszuschicken:

Zeichen	Bedeutung als Sonderzeichen	wörtlich
\	das nächste Zeichen wörtlich nehmen oder als Teil der Wortklammer \< \> oder als Teil von Rückbezugsoperatoren \1, \2, ... oder für kommandospezifische Abkürzungen \b, \w	\\ oder [\]
( )	Zusammenfassung von Ausdrücken	\(und \)
[ ]	Klammern der Bereichsangabe	\[und \]
\< \>	Klammern zur Abgrenzung von Worten	\\< und \\>
{ }	Häufigkeitsangabe für voriges Zeichen	\{ und \}
.	beliebiges Zeichen	\. oder [.]
+	voriges Zeichen muss mindestens einmal vorkommen	\+ oder [+]
*	voriges Zeichen darf beliebig oft vorkommen	* oder [*]
?	voriges Zeichen kommt Null oder ein mal vor	\? oder [?]
^	Zeilenanfang als erstes Zeichen der Bereichsangabe: Verneinung	\^
$	Zeilenende	\$ oder [$]
|	oder-Verknüpfung	| oder [|]
\1	Wiederholung des Inhalts der ersten Klammer	\\1

Aufgaben:

Benutzen Sie zum Lösen der folgenden Aufgaben das Kommando "**egrep**". Verwenden Sie dabei bitte nicht die "**-x**"- und "**-w**"-Optionen von "**egrep**" sondern die "regular expressions"-Ausdrücke.

II. 2.1. Finden Sie in der Datei "D" alle Zeilen, die mit groß oder klein geschriebenem "hier" beginnen.

II. 2.2. Finden Sie heraus, welche Zeilen der Datei "D" mit ":", "!" oder "?" enden.

II. 2.3. Finden Sie die Zeilennummern für alle Zeilen der Datei "D", welche mit "Die", "Der" oder "Das" beginnen und mit einem Punkt "." enden.

II. 2.4. Die harte Nuss:
Finden Sie in der Datei "D" alle Zeilen, bei denen am Zeilenanfang das gleiche Wort steht wie am Ende (nach dem letzten Wort soll noch ein Punkt stehen; Groß- bzw. Kleinschreibung soll in den beiden Worten identisch sein).

II. 2.5. Finden Sie die Zeilen der Datei "D", welche nur Ziffern und/oder Leerzeichen enthalten.

II. 2.6. Stotterkontrolle: Finden Sie alle Zeilen der Datei "D", die Zeichenketten enthalten, in welchen sich "Silben" (soll heißen: "mindestens drei aufeinander folgende Buchstaben") identisch wiederholen.

II. 2.7. Finden Sie in der Datei "D" alle Palindrome mit fünf Buchstaben (Palindrome sind Worte, die vorwärts und rückwärts gelesen identisch sind).

Lösungen:

II. 2.1. **egrep** *'^[hH]ier' D*

II. 2.2. **egrep** *'[:!?]$' D*

II. 2.3. **egrep -n** *'^D(ie | er | as).*[.]$' D*

II. 2.4. **egrep** *'^(\<[a-zA-Z]+\>).+\<\1\>[.]$' D*

II. 2.5. **egrep** *'^[0-9]+$' D*

II. 2.6. **egrep** *'([a-z]{3,})\1' D*

II. 2.7. **egrep** *'\<([a-z])([a-z])[a-z]\2\1\>' D*

Einheit 11

3 Suchen und verändern – "sed"

Ein Editor, bei dem man die Datei, welche man ediert, nicht einmal am Bildschirm sieht? Tiefste Steinzeit! Oder 'n Saurier? Vielleicht. Aber versuchen Sie 'mal, mit einem anderen Editor Informationen aus einer mehreren Gigabyte großen Datei zu extrahieren oder führen Sie doch ruhig 15 mal hintereinander die gleiche Serie von stupiden Textersetzungen in verschiedenen Dateien durch. Davon abgesehen: manchmal ist es besser, man sieht erst gar nicht, was man alles tut ...

3.1 Zur Arbeitsweise des "sed"

"**sed**" liest den zu verarbeitenden Text aus einer Datei und gibt den neu entstehenden Text auf den Bildschirm aus; die Ausgabe kann aber ganz problemlos durch ">" oder ≫ in eine Textdatei umgeleitet werden. Einfache Textverarbeitungskommandos kann man direkt in der Kommandozeile hinter "**sed**" eingeben, kompliziertere (d. h. mehrzeilige Anweisungen) schreibt man besser in eine kleine Textdatei und ruft diese dann mit "**sed**" auf. Das Kommando "**sed**" hat zwei wesentliche Vorteile: Erstens wird nicht die ganze Datei *nicht* in den Hauptspeicher des Computers geladen (besonders nützlich bei Monster-Dateien), und zweitens macht "**sed**" reichlich Gebrauch von "regular expressions". Leider stimmt das Repertoir nicht mit dem von "**egrep**" überein und einige Klammern müssen durch "\" eingeleitet werden:[1]

von "**sed**" erkannte Sonderzeichen:	[ ]	\	.	*	^	$
von "**sed**" nicht erkannte Sonderzeichen:	\< \>	|	+	?		
bei "**sed**" abweichende Klammern:	\{ \}	\(\)				

"Regular expressions" werden bei "**sed**" von zwei Divisionsstrichen "/" eingeklammert (z. B. sucht /[0-9]\{6\}/ nach sechs-stelligen Zahlen).

3.2 Zeilen löschen

Mit "**sed**" kann man entweder alle Zeilen aus einer Datei löschen, die einen bestimmten Suchbegriff enthalten, oder alle Zeilen, die in einem bestimmten Bereich liegen.

'Wo d'
'VonWo ,BisWo d'

An den Stellen "Wo", bzw. "VonWo" und "BisWo" kann entweder die Zeilennummer stehen oder eine "regular expression" (muss mit "/" eingeklammert sein!) oder das Zeichen für Dateiende "$". Zugegeben, die Syntax des "**sed**" ist etwas gewöhnungsbedürftig, aber die paar Optionen, die ich hier vorstelle, sind leicht zu merken.

sed *'1,10d' A>B* — löscht die ersten zehn Zeilen aus Datei "A" und schreibt die verbliebenen Zeilen in die Datei "B" ("**sed**" steht für "stream edit"; "d" steht für "delete").

[1] Da "**sed**" sowohl die geschweiften Klammern "{ }" in der Kommandosyntax als auch die Varinate "\{ \}" in "regular expressions" verwendet, bleibt nur noch die Möglichkeit "[{}]", wenn man mit "**sed**" nach "{" oder "}" sucht. Ähnliche Probleme dürften sich ergeben, wenn man nach "/" suchen möchte.

sed *'/Summe/,$d' A>B*	löscht alle Zeilen aus Datei "A" vom ersten Auftreten des Suchbegriffs "Summe" bis zum Dateiende und schreibt die verbliebenen Zeilen in die Datei "B".
sed *'/[a-zA-Z]/d' A>B*	löscht alle Zeilen von Datei "A", welche mindestens einen Buchstaben enthalten, und schreibt die verbliebenen Zeilen in die Datei "B".
sed *'/^[0-9]/d' A>B*	löscht alle Zeilen von Datei "A", welche mit einer Ziffer beginnen, und schreibt die verbliebenen Zeilen in die Datei "B".
sed *'/von/,/bis/d' A>B*	beginnend mit der Zeile, welche erstmals "von" enthält bis zur Zeile, welche danach erstmals "bis" enthält, werden Zeilen in der Datei "A" gelöscht und die verbliebenen Zeilen in die Datei "B" geschrieben.
sed *'/^$/d' A>B*	löscht alle leeren Zeilen aus Datei "A" und schreibt die verbliebenen Zeilen in die Datei "B". **Anmerkung:** Zeilen die Leerzeichen enthalten, sind keine leeren Zeilen.

3.3 Zeichenketten mit dem "sed" verändern

Ebenso wie für das vorige Kommando stehen auch für das Substitutionskommando wieder verschiedene Varianten zur Verfügung:

```
                's/AlteRegex/NeueRegex/[Option]'
          '[Wo]s/AlteRegex/NeueRegex/[Option]'
'[VonWo],[BisWo]s/AlteRegex/NeueRegex/[Option]'
```

Im ersten Fall gilt der Ersetzungsbefehl für alle Zeilen der Datei, im zweiten Fall nur für die angegebene Zeilennummer (bzw. für diejenigen Zeilen, die einen bestimmten Suchbegriff enthalten) und im dritten Fall gilt er nur im angegebenen Bereich. Wenn man keine "Option" angibt, wird in einer Zeile nur der erste gefundene Begriff "AlteRegex" durch "NeueRegex" ersetzt. Durch die Option "g" werden alle gefundenen Begriffe einer Zeile ersetzt ("g" steht für "global"). Durch Angabe einer Zahl kann man bestimmen, der wievielte gefundene Begriff einer Zeile ersetzt werden soll.

sed *'s/fehlt/-1/g' A>B*	ersetzt in Datei "A" alle "fehlt" durch "-1" und schreibt das Resultat in die Datei "B" ("s" steht für "substitute").
sed *'s/,/./g' A>B*	ersetzt in Datei "A" alle Kommata durch Punkte und schreibt das Resultat in die Datei "B". **Anmerkung:** Verblüffenderweise wird der Punkt in der zweiten "regular expression" wörtlich genommen und nicht als Sonderzeichen aufgefasst.
sed *'/^[0-9]/s/,/./g' A>B*	wie das vorige Kommando, jedoch wird die Ersetzung nur durchgeführt, wenn die Zeile mit einer Ziffer beginnt.

sed *'100,200s/,/./2' A>B* — wie das vorige Kommando, jedoch wird die Ersetzung nur von der 100. bis zur 200. Zeile durchgeführt und nur das zweite Komma in jeder Zeile wird ersetzt.

sed *'s/Pat//g' A>B* — entfernt aus Datei "A" alle "Pat" und schreibt das Resultat in die Datei "B".
Anmerkung: "Pat" wird hier durch *nichts* ersetzt, also gelöscht.

Natürlich unterstützt "**sed**" auch die Rückbezugsoperatoren "\1", "\2", ...

sed *'s/Nr\([0-9]*\)/\1/g' A>B* — löscht aus Datei "A" alle "Nr", welche von einer Zahl gefolgt werden, und schreibt das Resultat nach "B".
Anmerkung: Beachten Sie, dass sich "**sed**" nicht mit einfachen Klammern "( )" zufrieden gibt, sondern "\(\)" benötigt.

Wenn man, wie im obigen Beispiel, nur einzelne Zeichen ersetzen möchte, kann man auch das folgende Kommando anwenden:

```
                    'y/AlteZeichen/NeueZeichen/'
                'Wo'y/AlteZeichen/NeueZeichen/'
'VonWo','BisWo'y/AlteZeichen/NeueZeichen/'
```

sed *'y/,/./' A>B* — setzt in Datei "A" anstatt der Kommata Punkte und schreibt das Resultat in die Datei "B".

sed *'/[^a-zA-Z]/y/,./.,/' A>B* — vertauscht in Datei "A" in allen Zeilen, welche *keine* Buchstaben enthalten, Kommata und Punkte und schreibt das Resultat in die Datei "B".
Anmerkung: Die Ersetzung findet nicht sukzessiv sondern gleichzeitig statt (andernfalls würden alle neu gesetzten Punkte am Ende wieder zu Kommata zurückgesetzt).

3.4 Der Verneinungsoperator "!"

Hinter die Angabe "Wo" bzw. "VonWo,BisWo" kann ein "!" gesetzt werden, um anzugeben, dass (nur) in diesen Zeilen das nachfolgende Kommando *nicht* ausgeführt werden soll.

sed *'/[a-zA-Z]/!y/,./.,/' A>B* — identisch mit dem letzten Kommando aus dem vorigen Abschnitt: nur in denjenigen Zeilen, welche Buchstaben enthalten, werden Punkte und Kommata *nicht* vertauscht.

sed *'/^Gruppe/!d' A>B* — löscht in Datei "A" alle Zeilen *außer* denen, welche mit "Gruppe" beginnen, und schreibt die verbliebenen Zeilen in Datei "B".
Anmerkung: Dieses Kommando ist identisch mit: **sed** *'^Gruppe/p' A>B*

3.5 Skripte mit "sed" abarbeiten

Skripte haben in "**sed**" immer die folgende generelle Struktur, wobei "VonWo,BisWo" wie gewohnt durch "Wo" ersetzt oder völlig weggelassen werden kann:

```
VonWo,BisWo{
Kommando1
Kommando2
...
}
```

Wenn man nun noch beachtet, dass hinter den beiden geschweiften Klammern auf keinen Fall irgendetwas stehen darf (also auch keine Leerzeichen) und dass die schließende Klammer allein in einer Zeile stehen muss, kann eigentlich nicht mehr viel schief gehen. Nehmen wir an, wir haben das folgende Programm erstellt und unter dem Namen "skript" abgespeichert:

```
/^[0-9]/{       wähle alle Zeilen aus, welche mit einer Ziffer beginnen, und führe
y/.;,/ ./       in jeder dieser Zeilen die folgenden Befehle durch: (1) ersetze
s/")/2          Punkte und Strichpunkte durch Leerzeichen und Kommata durch
s/"/(/1         Punkte, (2) ersetze die zweiten Anführungszeichen durch ")" und
}               (3) ersetze die ersten Anführungszeichen durch "(".
```

Jetzt müssen wir nur noch "**sed -f** *skript A>B*" aufrufen, um in der Datei "A" diese Veränderungen durchzuführen.

Aufgaben:

Benutzen Sie zur Lösung der folgenden Aufgaben das Kommando "**sed**":

II. 3.1. Löschen Sie in Datei "A" alle Zeilen bis (einschließlich) zu der Zeile, welche erstmals den Begriff "Daten" enthält, und schreiben Sie das Resultat in "B".

II. 3.2. Entfernen Sie in den Zeilen von Datei "A", welche mit einer Ziffer beginnen, sämtliche Buchstaben (auch "ä", "Ä", "ö", "Ö", "ü", "Ü" und "ß") und schreiben Sie das Resultat in "B".

II. 3.3. Entfernen Sie in den Zeilen von Datei "A", welche *nicht* mit einer Ziffer beginnen, sämtliche Buchstaben (auch "ä", "Ä", "ö", "Ö", "ü", "Ü" und "ß") und schreiben Sie das Resultat in "B".

II. 3.4. Die harte Nuss:
Löschen Sie in Datei "A" alle (nicht-leeren) Zeilen, die keine zwei verschiedene Zeichen enthalten (z. B. nur "..."), und schreiben Sie das Resultat in "B".

II. 3.5. Erstellen Sie ein Skript, dass "**sed**" anweist, von der dritten bis zur letzten Zeile eines Dokuments alle klein und groß geschriebenen "ä" in "ae", "ö" in "oe", "ü" in "ue" sowie "ß" in "ss" zu verwandeln.

Lösungen:

II. 3.1. **sed** *'1,/Daten/d' A>B*

II. 3.2. **sed** *'/^[0-9]/s/[a-zA-ZäÄöÖüÜß]//g' A>B*

II. 3.3. **sed** *'/^[0-9]/!s/[a-zA-ZäÄöÖüÜß]//g' A>B*

oder: **sed** *'/^[^0-9]/s/[a-zA-Z]//g' A>B*

II. 3.4. **sed** *'/^\(.\)\1*$/d' A>B*

II. 3.5.
```
3,${
s/ä/ae/g
s/Ä/Ae/g
s/ö/oe/g
s/Ö/Oe/g
s/ü/ue/g
s/Ü/Ue/g
s/ß/ss/g
}
```

4 Wo liegt diese Datei bloß wieder 'rum? – "find"

Einheit 12

Zum Auffinden von Dateien bietet Linux das extrem mächtige Kommando "**find**", für das ich nur ein paar wenige von vielen möglichen Optionen vorstellen möchte. Sowohl für die Angabe der Dateinamen als auch für die Pfadnamen können die üblichen Wildcardzeichen (∗?) sowie die Bereichsangaben [] benutzt werden. Dateinamen müssen dann aber unbedingt von Anführungszeichen umgeben sein.

4.1 Suche nach Dateinamen

find -name *'∗.txt'*	sucht vom aktuellen Verzeichnis aus durch alle Unterverzeichnisse nach Dateien, die auf ".txt" enden.
find -iname *'∗.txt'*	wie voriges Kommando, aber ohne Berücksichtigung von Groß- und Kleinschreibung ("i" steht für "ignore case").

4.2 Beschränke die Suche auf spezielle Verzeichnisse

Die Suche nach Dateien lässt sich auch auf ganz bestimmte Verzeichnisse einschränken:

find *Verz[AB]* **-iname** *'∗.txt'*	sucht in den Verzeichnissen "VerzA" und "VerzB" nach Dateien mit der Endung ".txt".
find *Verz[AB]* **-follow -iname** *'∗.txt'*	sucht wie das vorige Kommando in den Verzeichnissen "VerzA" und "VerzB" nach Dateien mit der Endung ".txt", jedoch werden auch Querverweise ("links") weiter verfolgt.
find *VerzA* **-name** *'∗.doc'* **-maxdepth** *2*	wegen der Option "-maxdepth 2" sucht "**find**" nur im aktuellen Verzeichnis sowie in allen Unterverzeichnissen, die direkt im aktuellen Verzeichnis liegen (diese haben relativ zum aktuellen Verzeichnis die "Tiefe 2"), nach Dateien, welche auf ".doc" enden.
find *VerzA* **-name** *'∗.doc'* **-mindepth** *2* **-maxdepth** *2*	ähnlich, wie der vorige Befehl, aber diesmal ist die Suche auf alle Unterverzeichnisse eingeschränkt, welche direkt im Verzeichnis "DatenVerz" liegen, denn nur für diese gilt "mindepth ≤ depth ≤ maxdepth" – das Verzeichnis "VerzA" selbst wird nicht durchsucht.

4.3 Beschränke die Suche nach dem Dateialter

Es kommt noch besser: wir können auch nach allen Dateien suchen, die in den letzten Tagen (oder Minuten) verändert (oder nicht verändert) wurden:

find *VerzA* **-ctime** *-14* **-name** *'∗'.doc*	sucht im Verzeichnis "VerzA" nach allen Dateien, die auf ".doc" enden und *innerhalb der letzten 14 Tage* verändert wurden ("ctime" steht für "change time"). **Anmerkung:** Ebenso kann man auch die Option "**-cmin**" benutzen und die Zeit in Minuten angeben.
find *VerzA* **-ctime** *14* **-follow**	sucht im Verzeichnis "VerzA" (und allen Unterverzeichnissen, einschließlich der Querverweise) nach allen Dateien, die *vor genau 14 Tagen* zuletzt verändert wurden. **Anmerkung:** Ebenso kann man auch die Option "**-cmin**" benutzen und die Zeit in Minuten angeben. **Anmerkung:** Die Option "**-name** '∗'" kann entfallen).

find *VerzA* **-ctime** *+14* **-follow**	wie das vorige Kommando, jedoch werden alle Dateien gesucht, die *vor mehr als 14 Tagen* zum letzten mal verändert wurden. **Anmerkung:** Ebenso kann man auch die Option "**-cmin**" benutzen und die Zeit in Minuten angeben.
find *VerzA* **-cnewer** *'BriefAlt'* **-iname** *'*brief*'*	suche im Verzeichnis "VerzA" nach irgendwelchen Dateien, deren Namen (groß oder klein geschrieben) den Ausdruck "brief" enthalten und gib die Dateinamen auf den Bildschirm aus, *wenn die Dateien jünger sind als* die Datei "BriefAlt".

4.4 Finde und handle

Jetzt wird's erst richtig brutal: wir suchen nicht nur irgendwelche Dateien, sondern wir verwursten sie auch gleich:

find *VerzA* **-name** *'*.txt'* **-exec mv -v** {} V2 \;	sucht im Verzeichnis "VerzA" nach allen Dateien mit der Endung ".txt" und verschiebt sie in das Verzeichnis "V2". **Anmerkung:**: Eingeklemmt zwischen "-exec" und "\;" steht ein auszuführendes Kommando, bei dem der Name der Datei, mit der etwas geschehen soll, durch ein Paar geschweifte Klammern ersetzt wurde. Dieses Kommando wird für jeden "Treffer" von "**find**" ausgeführt ("exec" steht für "execute command").
find *VerzA* **-name** *'*.sik'* **-exec head** -1 {} \; **-exec rm -iv** {} \;	sucht im Verzeichnis "VerzA" nach allen Dateien mit der Endung ".sik", gibt für jede gefundene Datei die erste Zeile auf dem Bildschirm aus und fragt nach, ob sie gelöscht werden soll. **Anmerkung:** Für jede gefundene Datei werden zuerst alle Kommandos ausgeführt, bevor "**find**" weitersucht.
find *VerzA* **-name** *'*.dat'* **-ok mv -v** {} VerzB \;	sucht im Verzeichnis "VerzA" nach allen Dateien mit der Endung ".dat" und verschiebt sie in das Verzeichnis "VerzB". **Anmerkung:** Im Gegensatz zu "-exec" fragt "-ok" vor jeder Handlung nach. Nur wenn der Benutzer "y" eingibt, wird das Kommando mit der jeweiligen Datei ausgeführt.

4.5 Die maßgeschneiderte Suche

Leute, dieses Programm ist ein Monster: Optionen lassen sich gruppieren (durch "\(" und "\)"), verneinen (durch "!") und mit "-o" (nicht-exklusives "oder") verknüpfen:

find ! **-name** *'*.txt'*	sucht vom aktuellen Verzeichnis aus in allen Unterverzeichnissen nach Dateien, deren Namen *nicht* auf ".txt" enden.

find -name *'*.txt'* **-o -name** *'*.tex'*	sucht vom aktuellen Verzeichnis aus durch alle Unterverzeichnisse nach Dateien, deren Namen auf ".txt" *oder* ".tex" enden.
find -name *'*.sik'* **-o \(-name** *'*.txt'* **-ctime** *-30* **\)**	sucht vom aktuellen Verzeichnis aus durch alle Unterverzeichnisse nach Dateien, deren Namen auf ".sik" enden sowie nach Dateien, deren Namen auf '.txt' enden und die seit mindestens 30 Tagen nicht mehr geändert wurden.

Aufgaben:

Lösen Sie die folgenden Aufgaben mit dem Kommando "**find**":

II. 4.1. Was tut das folgende Kommando?
find *Verz[1-3]* **!** **\(-name** *'Dat.[0-9]'* **-o -name** *'Dat.[0-9][0-9]'* **\)**
-exec rm -v {} \;

II. 4.2. Was tut das folgende Kommando?
find *Briefe* **-iname** *'*.sik'* **-ctime** *-90* **-maxdepth** *1* **-ok mv** {} BrandNeu \;

II. 4.3. Suchen Sie im Verzeichnis "VerzA" (und dessen Unterverzeichnissen) nach allen Dateien, die weder auf ".jmp" noch auf ".xls" enden (ungeachtet der Groß- und Kleinschreibung der Endung).

II. 4.4. Löschen Sie in Ihrem Arbeitsverzeichnis und dessen Unterverzeichnissen alle Dateien namens "Schrott" und lassen Sie sich jeden Löschvorgang am Bildschirm anzeigen.

II. 4.5. Die harte Nuss:
Finden Sie alle Dateien namens "prog.pas", die älter sind als (oder gleich alt wie) die gleichlautende Datei in ihrem Arbeitsverzeichnis. Lassen sich von jeder gefundenen Datei die erste Zeile auf den Bildschirm ausgeben.

II. 4.6. Die harte Nuss:
Finden Sie in Ihrem Arbeitsverzeichnis und dessen Unterverzeichnissen alle Dateien, die älter als 60 Minuten sind, und benennen Sie diese Dateien um, indem Sie die Endung ".old" an den jeweiligen Dateinamen anhängen. Lassen Sie sich alle Umbenennungen am Bildschirm anzeigen.

Lösungen:

II. 4.1. Das Kommando löscht alle Dateien in den Verzeichnissen "Verz1", "Verz2" und "Verz3" und deren Unterverzeichnissen (falls vorhanden), außer den Dateien, deren Namen aus "Dat." und ein oder zwei Ziffern bestehen (ohne nachzufragen). Jede Löschaktion wird auf dem Bildschirm angezeigt.

II. 4.2. Das Kommando durchsucht das Verzeichnis "Briefe" *ohne* dessen Unterverzeichnisse nach Dateien, in deren Namen auf ".sik" (klein oder groß geschrieben) enden. Alle auf dieses Muster passenden Dateien, die innerhalb der letzten 90 Tage geändert wurden, werden in das Verzeichnis "BrandNeu" verschoben. Vor jedem Verschieben fragt der Computer aber noch einmal nach, ob diese auch wirklich erwünscht ist.

II. 4.3. **find** *VerzA* ! **-iname** *'*.jmp'* ! **-iname** *'*.xls'*

II. 4.4. **find -name** *'Schrott'* **-exec rm -v** {} \;

II. 4.5. **find -name** *'prog.pas'* ! **-cnewer** *'prog.pas'* **-exec head** -1 {} \;

II. 4.6. **find -cmin** *+60* **-exec mv -v** {} {}.old \;

5 Hilfe!

Einheit 13

5.1 Wer bin ich - und wenn "ja", wie viele?

Beginnen wir zunächst mit den philosophischen Grundfragen – der Frage nach der eigenen Identität, nach der Gesellschaft, sowie mit den Fragen nach Raum und Zeit:

whoami	Wer bin ich? Dieses Kommando gibt den Benutzernamen aus. Dies ist extrem wichtig, wenn man mehrmals unter verschiedenen Namen eingelogt ist – womöglich mit "super user"-Rechten.
pwd	Wo bin ich? Dieses Kommando gibt den Pfad zu meiner aktuellen Position an ("**pwd**" steht für "print working directory").
uname -a	gibt Informationen zum System aus: Betriebssystem, Name des Rechners, Datum und Uhrzeit sowie Prozessortyp ("a" steht für "all").

5.2 Und wer ist sonst noch alles da?

Ist diese Maschine wirklich so schnarchlangsam oder missbrauchen die lieben Kollegen mal wieder meine Kiste als Rechenknecht?

w	Wer ist sonst noch da? "**w**" gibt die Namen aller angemeldeten Benutzer aus und zeigt, wie lange jeder angemeldet ist, und wieviel CPU-Zeit er oder sie bereits verbraten hat.
finger *eichner*	"eichner"? Wer ist das? Dieses Kommando gibt weitere Informationen zu einem Benutzer aus. Jeder Benutzer kann übrigens zwei Textdateien ".plan" und ".project" in sein Heimatverzeichnis legen, die von "**finger**" auf dem Bildschirm ausgegeben werden.

5.3 Wie spät ist es? – "date" and "cal"

Zur Ausgabe von Zeit und Datum verwendet man

date	Ist heute nicht Sonntag? Dieses Kommando zeigt Wochentag, Datum und Uhrzeit an.
cal	gibt einen Kalender für den jetztigen Monat aus.
cal *7 2001*	gibt einen Kalender für Juli 2001 aus.
cal -y	gibt einen Kalender für dieses Jahr aus ("y" steht für "display entire year").

Nachdem jetzt unsere Existenz sowie unsere Position in Zeit, Raum und Gesellschaft geklärt ist, werden wir uns den praktischen Fragen des Computeralltags widmen.

5.4 Read the manual! – "man", "type", "apropos" and "whatis"

Meine am häufigsten benutzte Quelle für schnelle Informationen sind die "manual pages". Nach Aufruf von "**man**" erhält man – wie bei "**less**" eine Bildschirmausgabe, in der man mit den Cursorbewegungstasten ↑ und ↓ sowie mit den Bild↑– und Bild↓– vor- und zurückblättern kann. Zum Verlassen des Programms gibt man "q" ein.

man *rm*	*Ausführliche Hilfe:* Dieser Befehl gibt die "manual page" zum Kommando "**rm**" auf den Bildschirm aus, in der alle möglichen Optionen zum "**rm**"-Kommando erklärt werden. Für viele Optionen gibt es zwei verschiedene Formen: So ist z. B. "**rm -r** *Verz*" identisch mit "**rm --recursive** *Verz*" ("**man**" steht für "manual page").
type *la*	durch dieses Kommando erfährt man, dass sich hinter dem Kommando "**la**" in Wirklichkeit der Befehl "**ls -la** " verbirgt, also ein "**alias**" (für solche Befehle gibt es keine "manual page").
type *less*	für Standardlinuxkommandos erhält man den Ort, an dem sich die ausführbare Datei befindet.
apropos *pwd*	Dieser Befehl gibt für jeden Eintrag in einer "manual page", in welcher der Suchbegriff "**pwd**" auftaucht, eine einzeilige Beschreibung auf den Bildschirm aus; *per default* sucht "**apropos**" nach einer "regular expression".
whatis *cat*	*Kurzhilfe:* wie voriges Kommando – "**whatis**" sucht jedoch nicht nach der "regular expression" "**cat**" sondern nur nach dem Wort "**cat**".

Aufgaben:

Lösen Sie mit den in dieser Einheit gelernten Kommandos die folgenden Aufgaben:

II. 5.1. Zeit zum Mittagessen – wie spät ist es?

II. 5.2. Mit welchem Wochentag begann unser Kalender (d. h. auf welchen Wochentag fiel der 1.1.1)?

II. 5.3. Fallen Weihnachten und Neujahr dieses Jahr wieder 'mal auf's Wochenende?

II. 5.4. Die harte Nuss:
Auf welchen Wochentag fiel der 10.10.1752?

II. 5.5. Welcher Befehl wird durch Eingabe von "**md**" wirklich ausgeführt?

II. 5.6. Ich möchte, dass "**ls**" die Dateien der Größe nach sortiert. Wie geht das? rtm!

II. 5.7. "**lprm**"? Kenn' ich irgendwoher. Was war das doch gleich?

II. 5.8. In welchen "manual pages" finden sich Hinweise zu "**ln**"?

II. 5.9. Welche "manual pages" nehmen ausdrücklich Bezug auf die Programmiersprache "C" bzw. "C++"? Welche nehmen Bezug auf "Pascal"?

Lösungen:

II. 5.1. **date**

II. 5.2. **cal** *1 1*

II. 5.3. **cal** *7 2001*

II. 5.4. **cal** *10 1752*
Das kommt ganz darauf an, in welchem Land ...
In der Gregorianischen Kalenderreform wurden 10 Tage im Kalender gestrichen. Dieser Eingriff wurde am 4. Oktober 1582 durchgeführt. Im Laufe der folgenden vier Jahrhunderte zog ein Land nach dem anderen nach und passte seinen Kalender an. Der Programmierer von "**cal**" entschied sich dafür, das Loch zwischen dem 2. und 14. September 1752 zu setzen – mit der Begründung, dass bis dahin die Mehrheit der Länder sich der Kalenderreform angeschlossen hatte. Weitere Informationen erhalten Sie durch "**man** *cal*".

II. 5.5. **alias** *md*

II. 5.6. In der manual page nachsehen: "**man** *ls*"
Lösung: "**ls -S** " oder "**ls -Sr** "

II. 5.7. **whatis** *lprm*

II. 5.8. unnötig viele Stellen erhält man durch: **apropos** *'ln'*
am besten gefällt mir persönlich: **apropos** *'\<ln\>'*
zu wenige Stellen erhält man durch: **whatis** *ln*

Nach der Beschreibung der Befehle **apropos** und **whatis** hätte es zwischen der zweiten und dritten Lösung allerdings keinen Unterschied geben sollen. :-)

II. 5.9. **apropos** *'\<C\>'*
apropos *'\<Pascal\>'*

Leider arbeitet "**apropos**" mit "regular expressions", welche keinen Unterschied zwischen Groß- und Kleinschreibung zulassen, so dass in die Ergebnisliste für "C" auch ein Kommando ("**lyx**") geraten ist, das mit "C" nichts zu tun hat.

Einheit 14

6 Sag's 'mal anders – "alias"

alias *md*	durch dieses Kommando erfährt man, welcher Befehl nach "**md**" wirklich ausgeführt wird (zu "**md**" gibt es keine "manual page", da es sich um ein "**alias**" handelt).
alias	ohne weitere Angabe von Kommandos oder Optionen erhält man eine Liste aller "**alias**"-Einträge.
alias *tot='rm -rf'*	definiert ein neues "**alias**" – jetzt kann man anstelle von "**rm -rf**" auch "tot" verwenden.
unalias *tot*	löscht den Eintrag "tot" in der "**alias**"-Tabelle – "tot" steht nun nicht mehr für "**rm -rf**".

7 Was läuft da wieder alles? "ps", "pstree" und "top"

Heute 'mal wieder schnarch-langsam diese Kiste – und so 'was nennt sich "state-of-the-art"! Wahrscheinlich haben meine lieben Kollegen mich wieder einmal mit Prozessen eingedeckt. Also – wer hat da wieder seine Finger drin?

ps -u gibt eine Liste der momentan laufenden Prozesse an. Eine Auflistung der wichtigsten Ausgabespalten ist in der folgenden Tabelle gegeben ("**ps**" steht für "process status"; "u" steht für "include user name" und führt auch sonst zur Ausgabe von mehr Informationen).

USER	Wer hat den Prozess gestartet?			
PID	Was ist die Prozessnummer?			
%CPU	Wie stark ist die CPU durch diesen Prozess ausgelastet?			
%MEM	Wie stark ist der Speicher durch diesen Prozess ausgelastet?			
STAT	Welchen Status hat der Prozess?			
	R	running	S	asleep
	N	positive nice value	D	asleep and not interruptible
	T	stopped	Z	zombie
START	Seit wann läuft der Prozess?			
TIME	Wie lange läuft der Prozess schon?			
COMMAND	Wie heißt der Prozess?			

pstree -G zeigt die Prozesse graphisch in einem Stammbaum an ("**pstree**" steht für "process status tree" und ist kein Standard-Linuxprogramm).

Leider zeigen "**ps**" und "**pstop**" nur die lokal gestarteten Prozesse und nicht diejenigen, die mir über "remote login" beschert wurden. Das Kommando "**top**" ist da schon etwas kooperativer:

top -c zeigt alle momentan laufenden Prozesse an (vgl. obige Tabelle) und passt die Anzeige in regelmäßigen Abständen an die Veränderungen an ("c" steht für "command line display").
Anmerkung: Zum Beenden der Anzeige, tippen Sie "q".

top -p *1743* wie voriges Kommando, jedoch wird nur der Prozess mit der angegebenen Prozessnummer (hier "1743") angezeigt und verfolgt ("p" steht für "process ID").

8 Vordergrund und Hintergrund: "fg" and "bg"

mirrordir . */sik* **&** — das vor & stehende Kommando wird im Hintergrund durchgeführt. In diesem Beispiel wird ein Spiegelbild des momentanen Verzeichnisses und aller Unterverzeichnisse im Hintergrund angelegt.

Man kann einen bereits laufenden Prozess auch nachträglich in den Hintergrund "schieben". Dazu hält man den Prozess zunächst mit "Crtl c" an und gibt dann "**bg**" ein ("**bg**" steht für "background"). Will man nicht den zuletzt angehaltenen Prozess in den Hintergrund befördern, so muss man erst einmal die Prozessnummer des gestoppten Prozesses ermitteln und gibt dann den folgenden Befehl ein:

bg *1794* — schiebt den Prozess mit der angegebenen Prozessnummer (hier: "1794") in den Hintergrund.

kill *1734* — beendet den Prozess mit der angegebenen Nummer (hier "1734") und entfernt ihn aus dem Speicher.

fg *1794* — schiebt den Prozess mit der angegebenen Prozessnummer (hier: "1794") in den Vordergrund ("**fg**" steht für "foreground").

9 Liebe, hartnäckige und böse Prozesse: "nice" "nohup" and "kill"

Zunächst einmal zu den "lieben" Prozessen: Wenn mehrere rechenintensive Programme im Hintergrund laufen und ich nebenbei noch programmieren möchte, bin ich darauf angewiesen, dass sich diese Hintergrundprozesse "nett" verhalten und möglichst nur dann Rechenzeit anfordern, wenn der Computer sowieso gerade auf meine nächste Eingabe wartet (was natürlich mehr als 99.9% der Zeit der Fall ist).

nice -n *15 prog.exe* — startet das Programm "prog.exe" mit einem "nice level" von 15 (d. h. das Programm "prog.exe" verhält sich dadurch "unaufdringlicher").
Anmerkung: Das Standard-"nice level" ist auf 10 Punkte festgesetzt, das Maximum ist 20, das Minimum 1. Angaben unter 10 akzeptiert der Computer aber nur bei priviligierten Benutzern.

Leider stürzen selbst im Hintergrund gestartete Prozesse meistens ab, wenn ich mich am Computer abmelde, so dass es nötig ist, diesen Programmen etwas mehr "Hartnäckigkeit" zu geben:

nohup *prog.exe* **&** — startet das Programm "prog.exe" im Hintergrund und sorgt dafür, dass dieses Programm auch dann nicht beendet wird, wenn ich mich abmelde ("**nohup**" steht für "no hang-up").
Anmerkung: Für die Ausgabe des Programms "prog.exe" wird automatisch eine Datei "nohup.out" angelegt.

Manchmal hat man leider Programme gestartet, die nur Unfug machen, oder auf deren Beendigung man nun doch nicht mehr warten will oder kann. Damit solche Prozesse weder Speicher noch CPU weiterhin unnötig belasten, müssen sie entfernt werden. Dazu besorgt man sich zunächst die Prozessnummer und gibt dann das

folgende Kommando ein:

kill *1734*	beendet den Prozess mit der angegebenen Nummer (hier "1734") und entfernt ihn aus dem Speicher.
kill *-9 1734*	wie der vorige Befehl, jedoch mit etwas mehr "Nachdruck" für Prozesse, die dem vorigen "**kill**"-Kommando widerstehen.

Aufgaben:

sag' mal,

II. 9.1. was ist los?

Lösungen:

II. 9.1. ich weiß nicht

Einheit 15

Teil III
Einige ausgewählte Programme

1 Der Kommandozeileneditor "vi"

"**vi**" (gesprochen "wie Ei") ist der klassische Kommandozeileneditor ("**vi**" steht für "visual"). Es gibt mehrere Klone, die in etwa die gleichen Befehle verwenden, z. B. "**elvis**", "**nvi**", "**vile**" und "**vim**". Die hier verwendete Variante "**vim**" (steht für "vi improved") wurde mit der Bitte zur Verfügung gestellt, AIDS-Waisen in Uganda zu unterstützen.

1.1 Why "vi"?

Warum ausgerechnet "**vi**"? Auf den ersten Blick erschien mir dieses Programm mit seinen obstrusen Tastenkürzeln wie ein Rückfall in die tiefste Steinzeit – ersonnen von Asketen und Masochisten. Bei der Vorbereitung diese Einheit revidierte sich meine Ansicht drastisch: die Kürzel, die mir vorher als Atavismen erschien, erwiesen sich als gnadenlos logisch aufgebaut. Der Editor selbst zeigte eine Funktionalität, welche die Möglichkeiten moderner Maus-Klick-Programme meilenweit hinter sich lässt. Auch dann wenn Sie Ihren gewohnten Editor beibehalten wollen, bieten "**vi**"-Kenntnisse einen riesigen Vorteil: egal ob man sich über abenteuerliche Internetverbindungen angemeldet hat, oder ob das System fast völlig abgeschmiert ist und man über eine externe Verbindung Wiederbelebungsversuche macht – solange überhaupt noch irgendetwas funktioniert, funktioniert auch der "**vi**" noch.

1.2 "vi" starten und beenden

Also zuerst einmal das Programm anwerfen. Das geht ganz einfach mit

vi *Brief.dat* — öffnet die Datei "Brief.dat" mit dem "**vi**"-Editor. Falls noch keine solche Datei existiert, wird beim ersten Abspeichern des Textes eine angelegt.

Zur Orientierung ist es oft hilfreich, sich die aktuelle Position als Zeilen-Spalten-Koordinate angeben zu lassen:

Crtl **G**	gibt am rechten unteren Rand die Zeilen- und Spaltenkoordinaten für die jeweils aktuelle Cursorposition an.
:set nu Return	zeigt am linken Rand die Zeilennummern an.
:set nonu Return	schaltet die Anzeige der Zeilennummern wieder ab.

Soweit ganz gut, aber wir wollen's nicht gleich übertreiben – wie kommt man hier wieder raus? Ganz einfach: man versichert sich, dass man "**vi**" im Kommandomodus ist (im Zweifelsfall "Esc" drücken), und gibt das folgende Kommando ein:

:q!	beendet die Bearbeitung der Datei ohne Änderungen zu speichern ("q" steht für "quit").
:wq	speichert die Änderungen beendet die Bearbeitung der Datei ("w" steht für "write").
:w	speichert den aktuellen Stand der Datei.
:w *Br_neu*	speichert die Datei unter dem Namen "Br_neu".

1.3 Texte lesen

Zuerst einmal die gute Nachricht: Bei neueren Linux-Versionen kann man sich mit den Cursorbewegungstasten "[←]", "[→]", "[↑]", "[↓]", "[Pos1]", "[Ende]" und den Bildschirmtasten "[Bild↑]" und "[Bild↓]" durch das Dokument bewegen (Tastaturabkürzungen für den Notfall finden Sie im Abschnitt 1.10). Versuchen Sie auch einmal, zum "Durchblättern" eines Textes die Leertaste und "[Return]" zu benutzen. Die im Folgenden aufgeführte Liste von Abkürzungen für Cursorbewegungen mag völlig überladen erscheinen (obwohl sie nur eine kleine Auswahl ist). Es wäre auch völlig unangebracht, sich so viele Optionen zu merken, nur um den Cursor elegant zu verschieben. Wie in den folgenden Abschnitten deutlich wird, lassen sich diese Optionen jedoch mit anderen "**vi**"-Kommandos kombinieren und machen diese zu sehr brauchbaren Werkzeugen.

Zeichen	Cursorbewegung
:*15* [Return]	zum ersten Zeichen der 15. Zeile.
B	zum ersten Zeichen des vorigen Wortes ("B" steht für "back"). **Anmerkung:** "Worte" in diesem Sinne sind durch "whitespace" (Leerzeichen etc.) getrennte Textabschnitte.
W	zum ersten Zeichen des nächsten Wortes ("W" steht für "word", vgl. voriges Kommando).
(	zum ersten Zeichen des Satzes. **Anmerkung:** Das Satzende ist durch ".", "!" oder "?", gefolgt von zwei "whitespace"-Zeichen, definiert.
)	zum ersten Zeichen des nächsten Satzes.
{	zum ersten Zeichen des aktuellen Paragraphen.
}	zum ersten Zeichen des nächsten Paragraphen.
G	zum Ende der Datei.
17 |	zum 17ten Zeichen der aktuellen Zeile.
%	steht der Cursor auf einer Klammer ("( )", "[ ]" oder "{ }"), so wird nach der zugehörigen schließenden oder öffnenden Klammer gesucht (dazwischen gelegene Klammerpaare werden logisch aufgelöst und ignoriert). Andernfalls wird die nächste schließende Klammer gesucht.
/*Juni* [Return]	vor bis zum ersten Zeichen von "Juni" (vgl. Abschn. 1.7).
?*März* [Return]	zurück bis zum ersten Zeichen von "März" (vgl. Abschn. 1.7).

Zwei weitere Cursorbewegungen ("n" und "N") werden im Abschnitt 1.7 besprochen. Außerdem kann man die aktuelle Position des Cursors speichern, indem man eine Markierung setzt:

m*q*	setzt an die aktuelle Cursorposition eine (unsichtbare) Markierung und nennt sie "q".
`*q*	setzt den Cursor auf die als "q" markierte Stelle.

1.4 Texte schreiben und löschen

Bevor Sie nach dem Starten von "**vi**" die Tastatur berühren, sollten Sie den wichtigsten Befehl überhaupt kennen:

u macht den letzten Befehl rückgängig ("u" steht für "undo").

Mit den nachfolgenden Befehlen kommen Sie automatisch in den Schreibmodus und können einen Text eingeben. Man bleibt so lange im Schreibmodus, bis man ihn durch "Esc" beendet.

i Text vor der aktuellen Cursorposition einfügen ("i" steht für "insert").

R Text von der aktuellen Cursorposition an überschreiben ("R" steht für "retype").

cW das nächste Wort durch den eingegebenen Text ersetzen.
Anmerkung: Der Cursor sollte auf dem ersten Zeichen des Wortes stehen; eine ausführliche Bastelanleitung für "c"-Befehle gibt es im Abschnitt 1.5.

Wer nicht mit der "Entf"–Taste jedes Zeichen einzeln löschen möchte, kann das "delete"-Kommando "d" verwenden (mit "← Del" kann man leider nicht löschen). Eine ausführliche Bastelanleitung für "d"-Befehle gibt es im Abschnitt 1.5.
Vorsicht: "**vi**"-Kommandos können nur im Kommandomodus eingegeben werden. Um den Schreibmodus zu beenden, muss man "Esc" drücken.

dW löscht das Wort rechts vom Cursor ("dW" steht für "delete word").
Anmerkung: Der Cursor sollte auf dem ersten Zeichen des Wortes stehen.

d Ende löscht bis zum Ende der Zeile.

d% wenn der Cursor auf einer Klammer steht, so wird alles zwischen der markierten Klammer und der zugehörigen Klammer gelöscht (inklusive der beiden Klammern).

dd löscht die aktuelle Zeile komplett.

dG löscht alles von der aktuellen Zeile bis zum Ende des Dokuments.

1.5 Bastelanleitung für "vi"-Befehle

Multiplikatoren: Wann immer es irgendwie sinnvoll ist, kann man einem "**vi**"-Kommando eine Zahl voraussstellen, die dann als Multiplikator für dieses Kommando wirkt. So kann man z. B. "2 Bild↓" eingeben, statt zweimal die Taste "Bild↓" zu drücken. "10 →" bewegt den Cursor 10 Zeichen nach rechts, "5dd" löscht 5 Zeilen und "3u" macht die letzen 3 Befehle rückgängig.
Textbereich: Zu den drei Befehlen "delete" ("d"), "change" ("c") und "copy" ("y") muss noch eine Option angegeben werden, die den Bereich des Textes benennt, auf den sich das Kommando beziehen soll. Durch Angabe der geeigneten Option können Sie bestimmen, ob Sie z. B. das nächste Wort, die nächste Zeile oder den nächsten Absatz löschen wollen. Als Optionen können alle im Abschnitt

1.3 erwähnten Cursorbewegungen dienen (mit *Ausnahme* des mit Doppelpunkt beginnenden Kommandos, welches den Cursor an den Anfang einer bestimmten Zeile bewegt). So bewirkt "d10[←]", dass die 10 links vom Cursor gelegenen Zeichen gelöscht werden. "d3W" löscht die nächsten 3 Worte, "d[Ende]" löscht den Rest der Zeile und "dG" löscht den Rest der Datei. In diesem Zusammenhang ist es auch interessant, mit dem Kommando "m" eine Marke zu setzen (siehe Abschnitt 1.3). Hat man eine Marke namens "x" gesetzt, so löscht "d\`x" alles bis (vor oder zurück) zur Marke "x". Die Marke "x" wird durch das Löschen auch gleich mit entfernt.

Aktionsradius des Bereichs: Um den exakten Aktionsradius eines Kommandos zu verstehen, muss man sich vorstellen, dass sich (vor und nach der Cursorverschiebung) links vom Cursor eine Trennmarke befindet. Der von den beiden Trennmarken umschlossene Bereich wird dann gelöscht ("d"), ersetzt ("c") oder kopiert ("y").

Beispiel zum Aktionsradius: Nehmen wir einmal an, Sie wollen das Wort "`unver|w|üstlich`" bearbeiten, bei dem der Cursor auf dem "w" steht. Durch "3[→]" wird der Cursor auf das "t" verschoben und der Aktionsbereich umschließt den Bereich "wüs" von "`unver|wüs|t|lich`". Also löscht das Kommando "d3[→]" das markierte Zeichen "w" und die zwei rechts gelegenen Zeichen. Nehmen wir wieder an, dass der Cursor auf dem "w" steht. Durch "3[←]" wird der Cursor auf das "v" verschoben und der Aktionsbereich umschließt den Bereich "ver" von "`un|v|er|wüstlich`". Also löscht das Kommando "d3[←]" die drei links von "w" gelegenen Zeichen und lässt das markierte Zeichen "w" unversehrt.

Ganze Zeile bearbeiten: Das Lösch-, Ersetzungs- oder Kopier-Kommando kann auf die ganze Zeile angewandt werden, indem man den Kommandobuchstaben verdoppelt ("dd", "cc" und "yy"). Mit "yy" kopiert man die Zeile, in welcher der Cursor steht, und mit "5yy" kopiert man gleich 5 Zeilen.

Ausnahmen: Da das Kommando "%" den Cursor *auf* die zugehörige Klammer bewegt, müsste nach der obigen Regel die schließende Klammer in den Befehlen "y%", "c%" und "d%" ausgespart bleiben, was zwar konsequent aber meist unerwünscht wäre. Kurzum: in diesem Fall entschied man sich für den Pragmatismus. Ähnlich verhält es sich wohl mit der Option "G" (Dateiende): "dG" löscht nicht nur von der aktuellen Cursorposition bis zum Dateiende, sondern löscht die gesamte Zeile, in welcher der Cursor steht.

Einheit 16

1.6 Kopieren und einfügen

Der zuletzt (mit "d" oder "Entf") gelöschte Textabschnitt kommt in einen Zwischenspeicher und kann an einer beliebigen Stelle im Text eingefügt werden:

p	fügt die zuletzt kopierten oder gelöschten Textabschnitt rechts vom Cursor ein ("p" steht für "paste").

Mit dem Kommando "y" (steht für "yank") kann man einen Textabschnitt kopieren. In gleicher Weise wie "d" kann auch "y" mit den Cursorbewegungs-Optionen aus Abschnitt 1.3 kombiniert werden (siehe die Bastelanleitung in Abschnitt 1.5).

yy	kopiert die aktuelle Zeile.
y*3***W**	kopiert die nächsten 3 Worte. **Anmerkung:** Der Cursor sollte auf dem ersten Zeichen des ersten Wortes stehen.
y Pos1	kopiert den Text bis zum Zeilenanfang.
y Ende	kopiert den Text bis zum Zeilenende.
y*2***)**	kopiert den Text bis zum Ende des nächsten Satzes.
y}	kopiert den Text bis zum Ende des aktuellen Paragraphen.
y\`*q*	kopiert den Text bis zur Textmarke "q" (zuerst muss mit "**m***q*" eine Markierung gesetzt werden).

Der "**vi**" ermöglicht es auch, mehrere Zwischenspeicher gleichzeitig zu benutzen. Falls sich jemand dafür interessiert, soll er oder sie 'mal einen Blick in das Handbuch werfen.

1.7 Ausdrücke suchen

Die Suchfunktion von "**vi**" verwendet grundsätzlich "regular expressions". Leider werden von den in Teil II, Kapitel 2, beschriebenen "regular expressions" die Sonderzeichen "+", "?" und "|" nicht von "**vi**" unterstützt und die Klammern, welche einen Ausdruck zusammenfassen, müssen bei "**vi**" in der Form "\(" und "\)" eingegeben werden.

/^*R* Return	sucht von der aktuellen Position vorwärts nach einer Zeile, die mit "R" beginnt
?\<*kann*\> Return	sucht von der aktuellen Position rückwärts nach dem Wort "kann".
n	letzte Suche (mit "/" oder "?") wiederholen.
N	letzte Suche (mit "/" oder "?") in entgegengesetzter Richtung wiederholen.

Für die folgenden Befehle ist es wichtig, zu wissen, dass (sowohl beim Vorwärts- als auch beim Rückwärtssuchen) der Cursor auf das *erste* Zeichen des gefundenen Ausdrucks gesetzt wird.

d/; Return	löscht alles bis zum Auftreten des nächsten Semikolon (das ";" bleibt erhalten).

d?^*Neu* Return	löscht von der aktuellen Position alles zurück bis zum Anfang der letzten Zeile, welche mit "Neu" begann.
dn	wiederholt die letzte Suche und löscht alles zwischen der aktuellen Cursorposition und dem gefundenen Ausdruck.
d5n	wiederholt die letzte Suche 5 mal und löscht alles zwischen der aktuellen Cursorposition und dem zuletzt gefundenen Ausdruck.
c/, *dass* Return	sucht (vorwärts) nach ", dass" und ersetzt den Text zwischen der aktuellen Cursorposition und dem Komma des gefundenen Ausdrucks durch einen neu einzugebenden Text (die Eingabe wird mit " Esc " beendet).
y/*ist[.]*$ Return	kopiert alles zwischen der aktuellen Cursorposition und dem nächsten Ende einer Zeile, welches aus "ist." besteht.

1.8 Suchen und ersetzen"

Es ist höchste Zeit, Ihre "**sed**"-Kenntnisse aufzufrischen: der Ersetzungsbefehl von "**vi**" funktioniert nach den folgenden Mustern:

:*ZeilenBereich* **s/***SuchBegriff* **/***ErsetzungsBegriff*
:*ZeilenBereich* **s/***SuchBegriff* **/***ErsetzungsBegriff* **/**Option(en)

Das sollte Ihnen zumindest bekannt vorkommen: der erste Teil ist die übliche Adressangabe (die übrigens aus dem "**ex**"-Editor stammt, den sich "**vi**" einverleibt hat) und er zweite Teil ist die "**sed**"-Ersetzungssyntax (siehe Teil II, Abschnitt 3 auf Seite 51). Jeder Ersetzungsbefehl muss mit " Return " abgeschlossen werden (um in den Beispielen nicht unnötig Platz zu verschwenden, werde ich dies im weiteren voraussetzen).

Für den **Zeilenbereich** gibt es folgende Möglichkeiten:

1,10 das Kommando auf die Zeilen 1 bis 10 (einschließlich) beschränken.

11,**$** das Kommando auf die Zeilen 11 bis Dateiende beschränken.

. das Kommando auf die aktuelle Zeile beschränken.

Für die **Optionen** gibt es folgende Möglichkeiten:

g steht für "general": auch dann ersetzen, wenn das Suchmuster mehrmals in einer Zeile vorkommt.

c erst nach Rückfrage ersetzen.

Zur Auffrischung der Erinnerung noch ein paar Beispiele:

:.s/*ä***/***ae***/g**	ersetzt in der aktuellen Zeile alle "ä" durch "ae".
:*1***,$ s/***Murks***/***Schrott***/g**	ersetzt jedes Vorkommen des Begriffs "Murks" in der ganzen Datei durch "Schrott".
:*1***,***100***s/**^*d***/***D***/c**	durchsucht die ersten 100 Zeilen nach Zeilen, welche mit "d" beginnen, und fragt jedes mal nach, ob das initiale "d" durch ein "D" ersetzt werden soll.
:*100***,***200***s/,/[.]/g**	ersetzt von der 100-ten bis zur 200-ten Zeile alle Kommata durch Punkte. **Vorsicht:** "regular expressions"-Sonderzeichen!

:*1*,$ s/^*[0-9]*$*/	durchsucht die gesamte Datei nach Zeilen, die nur Ziffern enthalten, und ersetzt die gefundenen Zeilen durch Leerzeilen.

Eine wichtige Eigenschaft von "regular expressions", die noch nicht besprochen wurde, ist die Möglichkeit, Teilausdrücke zu zitieren. Um Teilausdrücke abzugrenzen verwendet man die Klammern "\(" und "\)". Wenn man z. B. den Namen "Ute \(Ott\)" sucht, so verändern die Klammern den Ausdruck selbst nicht, man kann jedoch den eingeklammerten Bereich durch "\1" zitieren: "\1" steht jetzt für "Ott".

:*1,10*s/*Ute \(Ott\)/Frau \1*/g	ersetzt in den ersten 10 Zeilen der Datei jedes Vorkommen von "Ute Ott" durch "Frau Ott".
:.s/*DM \([1-9][0-9]*\)/\1 DM*/g	ersetzt in der aktuellen Zeile jedes Vorkommen von Angaben wie z. B. "DM 100" in die üblichere Form "100 DM".

Die Schreibweise dieser Ersetzungsbefehle ist ziemlich gewöhnungsbedürftig, also üben wir die noch ein wenig. Genaugenommen ist "\1" nicht der Rückbezug auf den mit "\(" und "\)" eingeklammerten Ausdruck, sondern der Rückbezug auf den *ersten* der so eingeklammerten Ausdrücke. Wenn mehrere solcher Klammern existieren, kann man auch "\2", "\3" etc. benutzen:

:*1*,$ s/*\(ist\) \(dies\)/\2 \1*/g	verdreht jedes Vorkommen von "ist dies" in der ganzen Datei zu "dies ist".
:*1*,$ s/*^\([a-z]*\) \(.*\)$/\2\1*/g	sucht in der ganzen Datei nach Zeichenketten aus Kleinbuchstaben am Zeilenanfang und verschiebt diese ans jeweilige Zeilenende.
:*1*,$ s/*\([0-9]*\)[,.]\([0-9]\) cm /\1\2 mm*/g	ersetzt im ganzen Dokument Angaben in Zentimetern mit einer Nachkommastelle (z. B. "27,8 cm") durch die entsprechende Angabe in Millimetern (z. B. "278 mm".)

1.9 "vi" specials

Das Unangenehme an Kommandozeileneditoren ist, dass sie die Kommandozeile blockieren. Was ist aber nun, wenn man eben eine Datei umbenennen oder hierher kopieren muss? Zum Ausführen von Linuxkommandos gibt es zwei verschiedene Möglichkeiten:

Crtl Z	wechselt auf das Terminal ohne "vi" zu beenden (zurück kommt man durch Eingabe von "fg Return" auf der Kommandozeile).
:!*cp ~/dat .* Return	führt das auf ":!" folgende Linuxkommando aus (in diesem Beispiel den Kopierbefehl "cp *~/dat .*").

Zum Abschluss noch einige interessante Möglichkeiten, die bisher noch nicht erwähnt wurden:

:r*dat* Return	fügt die komplette Datei "dat" nach der aktuellen Zeile in die bearbeitete Datei ein.

1.10 Steinaxt statt Maus

Wenn schon 'mal 'was schiefgeht, dann aber gleich richtig: oft genug sind noch nicht einmal mehr die Cursortasten verfügbar und die Tastatur wird auch gleich auf "US keyboard" zurückgestellt. Um sich innerhalb der Datei bewegen zu können, sind die Cursorbewegungen und das Scrollen aber nun einmal von elementarer Wichtigkeit. Deshalb sind hier die normalerweise unnötigen "**vi**"-Tastenbelegungen aufgeführt:

h	Cursor 1 Zeichen nach links bewegen (wie "[←]").
l	Cursor 1 Zeichen nach rechts bewegen (wie "[→]").
k	Cursor 1 Zeichen nach oben bewegen (wie "[↑]").
j	Cursor 1 Zeichen nach unten bewegen (wie "[↓]").
0	Cursor auf das erste Zeichen der Zeile setzen (wie "[Pos1]").
$	Cursor auf das letzte Zeichen der Zeile setzen (wie "[Ende]").
[Crtl] **F**	1 Seite vorwärts blättern ("F" steht für "forward"; wie "[Bild↓]").
[Crtl] **B**	1 Seite zurück blättern ("B" steht für "backward"; wie "[Bild↑]").
x	vom Cursor markiertes Zeichen löschen (wie "[Entf]").

Aufgaben:

Öffnen Sie die Datei "Exercise.txt" mit dem "**vi**" und lösen Sie die folgenden Aufgaben mit dem entsprechenden "**vi**"-Kommando.

III. 1.1. [Die harte Nuss:]
Was tut das folgende "**vi**"-Kommando?
:*1*,**$ s/**\<\(*[Dd]a*\)ß\>**/**\1*ss*/**g** [Return]

Einheit 17

Teil IV
Frutti di mare: "shells"

Die von Ihnen eingegebenen Kommandos müssen vom Computer zuerst einmal interpretiert und in eine für den Rechner verdauliche Form gebracht werden. Diese Aufgabe übernimmt für Sie die "shell". Linux stellt verschiedene "shells" zur Verfügung. *Per default* bekommen Sie als Kommandozeileninterpreter die sogenannte "bash shell". Diese "shell" ist eine Weiterentwicklung der kommerziellen "sh" von Herrn Bourne – daher auch der Name: "bash" ist die Abkürzung der Verballhornung "Bourne-again-shell". Neben der "bash" stehen Ihnen noch die "csh" und deren verbesserte Version, die "tcsh" zur Verfügung. Die einzelnen "shells" unterscheiden sich in der Syntax, mit der Kommandos eingegeben werden müssen (z. B. in der Bedeutung und Verwendung von Wildcards und Sonderzeichen). Darüber hinaus gibt es einige "shell"-spezifische Kommandos, die von anderen "shells" nicht (oder zumindest anders) ausgeführt werden.

> "**bash** is considered a powerful programming shell, while scripting in **csh** is rumored to be hazardous to your health."
>
> Linux in a Nutshell, 4th ed., 2000

Leider gleicht dieser Abschnitt eher einem Steinbruch als einer fertig aufgebauten Einheit. Ich wollte mich in dieser Einheit über Shells und Shellskripte auslassen, bin aber nicht damit fertig geworden und werde es vermutlich auch nie werden.

1 Schnelle Kommandoeingaben in der "bash"

1.1 Kommandos aus der "bash history"-Liste zurückholen

Vermutlich haben Sie längst herausgefunden, dass man auf der Kommandozeile mit den Cursortasten ↑ und ↓ alte Kommandos zurückholen kann. Die von Ihnen eingegebenen Kommandos werden ind der Datei "~/.bash_history" abgespeichert. Die Anzahl der Einträge können Sie mit dem Kommando "**set**" bestimmen. Durch Eingabe von "**tail** *-20 ~/.bash_history*" können Sie sich die letzten 20 der abgespeicherten Kommandos ansehen. Da die Kommandos "gepuffert" in die Datei geschrieben werden, werden Ihre letzten Kommandos vermutlich nicht mit ausgegeben. Für die im folgenden erläuterten Operationen stehen sie dennoch zur Verfügung. Sie können auch das Kommando "**history**" eingeben, um eine *komplette* Liste der verfügbaren Kommandogeschichte zu erhalten. Um ein Kommando aus dieser Liste auszuwählen, haben Sie (unter anderem) folgende Möglichkeiten:

↑ ↓	zwischen den gespeicherten Kommandos hin und her wechseln.
Esc <	an den Anfang der "**history**"-Liste wechseln.
Esc >	an das Ende der "**history**"-Liste wechseln.
Crtl r	sucht die "**history**"-Liste von der gegenwärtigen Positon an rückwärts nach der Zeichenkette ab, die man nach dieser Tastenkombination angibt. Sobald man das gewünschte Kommando gefunden hat, kann man es entweder durch Return sofort abschicken oder Crtl eingeben und das Kommando anschließend edieren.

!*mirr*	sucht das letzte Kommando aus der "**history**"-Liste, das mit "mirr" begann (z. B. "**mirrordir -v** . */sicher*") und führt es aus (ohne nachzufragen).
!?*sich*	sucht das letzte Kommando aus der "**history**"-Liste, das die Zeichenkette "sich" enthält (z. B. "**mirrordir -v** . */sicher*") und führt es aus (ohne nachzufragen).

1.2 Kommandozeilen edieren

Natürlich haben Sie schon einmal Gebrauch davon gemacht, daß man sich den angefangenen Namen eines Kommandos oder einer Datei durch die Tab –Taste automatisch vervollständigen lassen kann, und Sie haben auch schon das eine oder andere "zurückgeholte" Kommando unter Verwendung der Cursortasten ← und → verändert. Damit kennen Sie bereits die wichtigsten Editorfunktionen der Kommandozeile. Diese Auswahl (aus Dutzenden verschiedener Möglichkeiten!) möchte ich hier noch um einige nützliche Funktionen erweitern (leider sind diese Kommandos von der jeweiligen Linux-Installation abhängig, so dass einige davon nicht überall funktionieren.)

Tab	versucht das angefangene Kommando oder den Dateinamen zu ergänzen. Falls es hierfür mehr als eine Möglichkeit gibt, erhält man nach zweimaligem Drücken eine Liste aller Ergänzungsmöglichkeiten.
Pos1	zum Anfang der Zeile wechseln.
Ende	zum Ende der Zeile wechseln.
Crtl _	macht die letzte Veränderung der Kommandozeile rückgängig.
Crtl l	Bildschirm außer der gerade bearbeiteten Zeile löschen (identisch mit Kommando "**clear**").

1.3 Kopieren und Einfügen auf der Kommandozeile

Sehr nützlich sind auch die folgenden "cut" und "paste" Befehle. Beachten Sie, dass die Trenngrenze immer *vor* der aktuellen Cursorposition liegt: beim Ausschneiden "links vom Cursor" bleibt das vom Cursor markierte Zeichen erhalten, während es beim Ausschneiden "rechts vom Cursor" mit entfernt wird.

Crtl k	löscht von der gegenwärtigen Position zum Zeilenende.
Crtl u	löscht von der gegenwärtigen Position zum Zeilenanfang.
Crtl w	löscht links vom Cursor bis zur vorigen Leerstelle.
Crtl y	fügt den ausgeschnittenen Text links vom Cursor ein.

1.4 Kommandozeilen wie mit "emacs" oder "vi" edieren

Die meisten der bisher beschriebenenen Tastenkürzel stammen aus dem "**emacs**"-Editor. In Teil III, Einheit 1, stelle ich Ihnen einige Kommandos des "**vi**"-Editors

vor. Geben Sie das folgende Kommando ein, wenn Sie möchten, dass sich die Kommandozeile so verhält, als befänden Sie sich im "**vi**":

set -o *vi*	stellt die Kommandozeile auf die Bearbeitung durch "**vi**"-Befehle um. Anfangs befindet man sich im Texteingabemodus (in den Kommandomodus wechselt man mit "Esc", zurück in den Texteingabemodus mit "i").
set -o *emacs*	stellt die Kommandozeile auf die Bearbeitung durch "**emacs**"-Befehle um (Grundeinstellung).

2 Kommandoformen der "bash"

2.1 Ausgabe eines Kommandos umleiten ">" und "≫"

Manchmal ist es nützlich, den Output, der normalerweise auf der Kommandozeile landet, in eine Datei umzuleiten.
Anmerkung: Nur der sogenannte "standard output" wird umgeleitet (dazu gehören z. B. nicht die Fehlermeldungen).

ll > *Liste*	schreibt das Ergebnis von "**ll**" direkt in die Datei "Liste". Falls bereits eine Datei "Liste" existiert, wird diese ohne Warnung überschrieben. Andernfalls wird eine neue Datei angelegt.
ll ≫ *Liste*	hängt das Ergebnis von "**ll**" an das Ende der Datei "Liste" an. Falls keine solche Datei existiert, wird eine angelegt.

2.2 Mehrere Kommandos gleichzeitig abschicken ";" und "(;)"

Mehrere Kommandos können – getrennt mit Semikolon(s) ";" – gemeinsam in eine Kommandozeile geschrieben werden. Dies ist besonders dann nützlich, wenn man lange auf die Ausführung des ersten Kommandos warten muss, oder wenn man plant, die gleiche Kombination mehrfach zu verwenden, und sie später wieder aus der "history"-Liste holen möchte. Durch Klammerung "( )" kann man mehrere Kommandos zu einer logischen Einheit zusammenfassen, was besonders im Zusammenhang mit den später zu besprechenden Kommandoformen interessant sein kann.

rm ∗*.dat* **; cd**	löscht alle Dateien, welche auf ".dat" enden, und wechselt ins Heimatverzeichnis.
(date ; w; pwd)	durch die Klammerung werden die drei Kommandos so behandelt als wären sie ein einziges Kommando.

2.3 Kommandosubstitution "$( )"

Durch Kommandosubstitution kann man den Output eines Kommandos so als Input für ein anderes Kommando verwenden, als hätte man ihn wörtlich dort hin geschrieben: Das Kommando, welches den Ersetzungstext liefert, muss von der Klammer "$( )" umgeben sein. Angenommen das Kommando "**ls** ∗*.txt*" liefert das Ergebnis "Masern.txt", dann ist das Kommando "**egrep -in** *'krank'* **$(ls** ∗*.txt***)**" identisch mit dem Kommando "**egrep -in** *'krank' Masern.txt*"

cat $(ls V1/∗.txt**)** > *Texte*	hängt all diejenigen Dateien des Verzeichnisses "V1" zusammen, welche auf ".txt" enden, und speichert das Resultat als Datei "Texte" ab.

rm -v $(find -iname *'murks'* **)**	sucht im momentanen Verzeichnis und allen Unterverzeichnissen nach Dateien, die (groß oder klein geschrieben) den Begriff "murks" enthalten, löscht diese Dateien und gibt das Resultat auf dem Bildschirm aus. **Anmerkung:** Dieser Befehl funktioniert nur deswegen, weil "**find**" die Dateinamen *mit* Pfadangaben ausgibt. **Anmerkung:** Dieses Kommando ist identisch mit dem Kommando "**find -iname** *'murks'* **-exec rm -v** {} \;".

2.4 Pipelines "|"

Durch die sogenannte "pipe" "|" wird der "Standardausgabekanal" des einen Kommandos mit dem "Standardeingabekanal" des anderen Kommandos zusammengelegt. In der Regel bedeutet dies, dass der Ausgabetext des vorausgehenden Kommandos als *Eingabetext* für das folgende Kommando verwendet wird (eine Ausnahme bilden Fehlermeldungen, da diese nicht über den "Standardausgabekanal" ausgegeben werden). Viele Kommandos, die Textdateien verarbeiten, erlauben es, den zu bearbeitenden Text über die "pipe" einzugeben (z. B. "**egrep**", "**wc**", "**less**", "**cat**", "**head**", "**tail -f**" etc.).

ls *Verz1* **| wc -l**	zählt die Anzahl der Dateien im Verzeichnis "Verz1" (die aus "**ls**" resultierenden Dateinamen werden als Text an "**wc**" übergeben). **Anmerkung:** Seltsamerweise werden die von "**ls**" ausgegebenen Dateinamen so interpretiert als stünden sie in verschiedenen Zeilen. **Anmerkung:** Unterverzeichnisse und Querverweise zählen ebenfalls als Dateien.
ls -R | egrep '^[0-9]+$'	durchsucht das aktuelle Verzeichnis samt aller Unterverzeichnisse nach Dateien, deren Namen nur aus Ziffern bestehen (die aus "**ls**" resultierenden Dateinamen werden als Text an "**egrep**" übergeben). Diese Befehlskombination erlaubt die Verwendung von "regular expressions" mit "**ls**"
cat $(ls V1/*.txt**) | egrep -i** *'Müller'*	wie das letzte Kommando von Abschnitt 2.3, jedoch wird der Inhalt der Textdateien mit "**egrep**" nach dem Begriff "Müller" (groß oder klein geschrieben) durchsucht und die gefundenen Zeilen ausgegeben (das Kommando "**ls**" gibt die gefundenen Dateinamen an "**cat**" weiter und dieses gibt den zusammengehängten Text all dieser Dateien an "**egrep**" weiter).

Index