DIHSÜD

FH | JOANNEUM
University of Applied Sciences

Linux Grundlagen

Manfred Pamsl

Über mich

- Manfred Pamsl

- Technische Mathematik / Informations- und Datenverarbeitung an der TU Graz
- Softwareentwicklung und IT Lösungen in der Telekommunikations-Branche
- Lehr- und Forschungsfokus
  - Cloud Computing
  - Server-Security

Inhalt

Einführung

Struktur des Dateisystems

Ein- und Ausgabe-Umleitung

Weitere Kommandos, Pipelines

vi(m) Editor

Einführung

Unix / Linux Eigenschaften

- „**Multi-Tasking**“
  - Das System kann mehrere Programme zur selben Zeit ausführen
- „**Multi-User**“
  - Das System unterscheidet zwischen verschiedenen Benutzern mit unterschiedlichen Rechten
- „**Multi-Session**“
  - Das System erlaubt die gleichzeitige Anmeldung („Session“) von mehreren Benutzern
  - Derselbe Benutzer kann mehrfach angemeldet sein

Login

- Ein Multi-User System erfordert **Anmeldedaten** beim Login
  - Z.B. mittels Benutzername und Password
  - Bestimmt die Authentisierung und damit die damit verbundene Autorisierung
  - Das Eingabe-Echo des Passworts wird maskiert oder vollständig unterdrückt
- Eine Session kann auf einem grafischen und Text-basierten Endgerät („Terminal“) gestartet werden
  - Lokale Text Konsole oder „Display Manager“
  - Über das Netzwerk mittels SSH, Remote Display Manager (über XDMCP), VNC, ...

Text Login

- Führt zum Aufruf eines interaktiven Kommandozeilen Interpreters („**Shell**“)
  - Es stehen verschiedene Shells zur Verfügung: bash, zsh, csh, ksh, ...
- Die Shell gibt eine Aufforderung („**Prompt**“) zur Kommando Eingabe aus
  - Endet üblicherweise mit $ für nicht privilegierte Benutzer und mit # für den System Administrator („root“)
  - Kann angepasst werden
- Die Ausführung des Kommandos startet nachdem die Eingabe Taste gedrückt wurde
  - Die Ausgabe des Kommandos erscheint vor der nächsten Eingabeaufforderung

Einfache Kommandos

- **id**
  - Zeigt die eigenen Benutzerdaten und Gruppenzugehörigkeiten an
- **pwd**
  - Zeigt das aktuelle Arbeitsverzeichnis
- **tty**
  - Zeigt das verwendete Endgerät („Terminal“, „teletype“) an
- **who**
  - Zeigt die aktiven Sessions aller Benutzer an
- **date**
  - Zeigt das aktuelle Datum und die Uhrzeit an

Allgemeine Kommando Syntax

- $ <command> <option(s)> <object(s)>
  - <command>: was wird ausgeführt
  - <option(s)>: wie wird es ausgeführt
  - <object(s)>: was ist vom Kommando betroffen
- *Options* und *Objects* sind oft optional
- Option `--help` oder `-h` gibt üblicherweise eine kurze Hilfe zur Kommando Syntax aus
- Beispiel:

```
$ id -u root
```

Bemerkungen zur Eingabe

- Alle Zeichen nach `#` werden als Kommentar behandelt und nicht von der Shell interpretiert
- **Kontrollzeichen**
  - `<STRG-c>` unterbricht die Kommando Eingabe und Ausführung
  - `<STRG-s>` Hält die Terminal Ausgabe an
  - `<STRG-q>` Nimmt die Ausgabe am Terminal wieder auf
  - `<STRG-z>` Stoppt die laufende Kommando Ausführung und legt es in den Hintergrund
    - Kann mit dem `fg` Kommando wieder in den Vordergrund geholt werden
  - `<STRG-d>` End-of-File Zeichen, beendet die laufende Shell

Auflisten eines Verzeichnis Inhalts

- **ls** Kommando listet („list“) den Inhalt eines Verzeichnisses auf
  - Option -l gibt eine detaillierte Auflistung der Eigenschaften, nicht nur den Namen
  - Option -a listet alle Objekte, auch jene, deren Name mit dem Zeichen . Beginnen
  - Option -d listet im Fall eines Verzeichnisses das Verzeichnis selbst und nicht den Inhalt
  - Die angegebenen Objekte spezifizieren, welche Verzeichnisse oder Dateien gelistet werden sollen

```
$ ls                    # Zeigt die Objekte im aktuellen Arbeitsverzeichnis
$ ls -l -a              # Detaillierte Auflistung aller Objekte im Arbeitsverzeichnis
$ ls -l /etc/passwd     # Detaillierte Anzeige der /etc/passwd Datei
$ ls -l /home           # Detaillierte Auflistung der Objekte im /home Verzeichnis
$ ls -l -d /home        # Detaillierte Informationen über das /home Verzeichnis selbst, nicht des Inhalts
```

Anzeige des Inhalts einer Datei

- **cat**
  - Zeigt den Inhalt einer oder mehrerer Dateien hintereinander („concatenate“) an
- **more** und **less**
  - Zeigt gibt den Inhalt einer Datei seitenweise an
- **head**
  - Zeigt die ersten (10) Zeilen einer Datei an
- **tail**
  - Zeigt die letzten (10) Zeilen einer Datei an

Online Hilfesystem

- Viele Kommandos zeige eine Hilfe zur Benutzung mittels der --help or -h Optionen an
- Aufruf des Online Handbuchs:

```
$ man ls        # Handbuchseite zum „ls“ Kommando
$ man -k 'working directory'      # Schlüsselwortsuche
```

Struktur des Dateisystems

Das „Root“ Verzeichnis /

- Es gibt nur einen logischen Dateisystem Baum der mit dem Wurzel („root“) Verzeichnis / beginnt
  - Alle Objekte (Dateien, Verzeichnisse, ...) sind über Pfade vom Root-Verzeichnis aus zugänglich
  - Es gibt keine Laufwerksbuchstaben
- Nur der Administrator kann den Inhalt des Root-Verzeichnis verändern
- Normale Benutzer können Dateien im Root-Verzeichnis nur auflisten und lesen

Hierarchische Struktur

- Jedes Verzeichnis kann Dateien oder weitere Unterverzeichnisse beinhalten
  - Dadurch können nahezu beliebig tiefe Strukturen entstehen
  - Sehr tiefe Strukturen können Probleme mit manchen Applikationen verursachen
- Spezielle Verzeichniseinträge
  - Der Eintrag `.` verweist auf das aktuelle Verzeichnis
  - Der Eintrag `..` verweist auf das übergeordnete Verzeichnis
  - `ls -a` zeigt diese Einträge in jedem Verzeichnis an

Namen von Dateisystem Objekten

- Namen können bis zu 255 Zeichen beinhalten
- Es können alle Zeichen bis auf ‚/‘ verwendet werden
  - **Metazeichen** der Shell wie Tabulator, Leerzeichen, *, ?, [,], -, $, sollten vermieden werden
- Namen sind „**case-sensitive**“
- Es gibt keine strukturellen Dateinamen Erweiterungen, ein Punkt im Namen ist nur ein normales Zeichen
  - Ausnahme: Objekte mit einem **Punkt am Anfang** werden üblicherweise bei der Auflistung nicht angezeigt

Einige Unterverzeichnisse von „/“

/bin
/boot
/dev
/etc
/home
/lib
/media
/mnt
/proc
/root
/sbin
/tmp
/usr
/var

Dateisystem Pfade

- Pfade spezifizieren den **Weg zu einem Objekt** im Dateisystem Baum
  - Verzeichnisebenen werden mittels ‚/' voneinander getrennt
- **Absolute Pfadnamen** beginnen mit ‚/'
  - Geben die Verzeichnisebenen beginnend mit dem **Root-Verzeichnis** an
- **Relative Pfadnamen** beginnen NICHT mit einem ‚/'
  - Geben die Position relativ zum aktuellen **Arbeitsverzeichnis** an

```
$ ls /usr/share/man    # absolut
$ ls /etc/passwd       # absolut
$ ls data              # relativ
$ ls ../../etc/passwd  # relativ
```

Dateisystem Navigation

- Das aktuelle Arbeitsverzeichnis kann mit pwd („print working directory“) abgefragt und mit cd („change directory“) verändert werden
- Nach dem Login ist das Arbeitsverzeichnis das „Home“ Directory des Benutzers
- Es können relative und absolute Pfade verwendet werden

```
$ pwd          # Zeige das aktuelle Arbeitsverzeichnis an
$ cd           # Wechsle in das eigene Benutzer („Home“) Verzeichnis
$ cd /etc      # Wechsle das Arbeitsverzeichnis auf ‚/etc‘
$ cd ..        # Gehe eine Ebene nach oben
$ cd ../../etc # Gehe zwei Ebenen nach oben, dann in ‚etc‘
$ cd /         # Gehe in das Root-Verzeichnis
```

Wichtige Dateisystem Operationen

- Erstellen eines Verzeichnis: **mkdir**
- Löschen von leeren Verzeichnissen: **rmdir**
- Kopieren von Dateien und Verzeichnissen: **cp**
- Verschieben / umbenennen von Dateien und Verzeichnissen: **mv**
- Löschen von Dateien und Verzeichnissen: **rm**

```
$ mkdir dir1 dir2              # Erstelle Verzeichnis dir1 und dir2
$ mkdir -p dirA/dirB/dirC      # Falls notwendig, erstelle zusätzlich die übergeordneten Verezichnisse dirA  und dirA/dirB (Option „-p“)
$ rmdir -p dirA/dirB/dirC      # Lösche auch die Verzeichnisse dirA and dirA/dirB falls sie leer sind (Option „-p“)
$ cp file1 file2               # Erstelle eine Kopie file2 von file1
$ cp file1 file2 file3 dir1    # Falls mehere Quelldateien angegeben werden muss das Ziel ein Verzeichnis sein
$ cp -r dir1 dir2 targetdir    # Kopiere die Verzeichnisse dir1 und dir2 rekursiv (Option „-r“) nach targetdir
$ rm file1 file2 file3         # Löschen der drei Dateien file1, file2 und file3
$ rm -r dir1 dir2              # Lösche rekursiv (Option „-r“) die Verzeichnisse dir1 und dir2
```

Übung: Dateisystem (1)

- Erstellen Sie in Ihrem Home-Directory ein Verzeichnis „übung1“
- Listen Sie zur Überprüfung den Inhalt Ihres Home-Directory
- Kopieren Sie das gesamte Verzeichnis „/tmp/exercise“ in das neue Verzeichnis „übung1“
- Überprüfen Sie das Ergebnis, indem Sie den Inhalt von „übung1“ und „übung1/exercise“ auflisten:

```
$ ls übung1
exercise
$ ls übung1/exercise
country.txt  dagger.txt  friends.txt  sighright.txt  sigh.txt  logs
```

Übung: Dateisystem (2)

- Zeigen Sie Ihr aktuelles Arbeitsverzeichnis mit `pwd` an
- Wechseln Sie mit `cd` in das Verzeichnis „übung1“
- Geben Sie ein detailliertes Listing aller Objekte in Ihrem Arbeitsverzeichnis aus
- Geben Sie ein detailliertes Listing aller Objekte im Unterverzeichnis „exercise“ aus
- Erstellen Sie in Ihrem Arbeitsverzeichnis „übung1“ ein Verzeichnis „backup“

Übung: Dateisystem (3)

- Kopieren Sie von Ihrem Arbeitsverzeichnis („übung1“) die Datei „country.txt“ im Verzeichnis „exercise“ in das Verzeichnis „backup“
- Wechseln Sie in das Verzeichnis „exercise“ und kopieren Sie die Dateien „dagger.txt“ und „friends.txt“ in das „backup“ Verzeichnis
- Wechseln Sie in das „backup“ Verzeichnis und kopieren Sie die noch fehlenden Dateien aus „exercise“ in Ihr Arbeitsverzeichnis
- Überprüfen Sie, dass die Verzeichnisse „exercise“ und „backup“ die gleichen Dateien beinhalten

Übung: Dateisystem (4)

- Wechseln Sie in das Verzeichnis „exercise“
- Löschen Sie die Datei „country.txt“ aus dem Arbeitsverzeichnis
- Stellen Sie die Datei mit Hilfe der Kopie im „backup“ Verzeichnis wieder her
- Löschen Sie mit dem Kommando `rm *.txt` alle auf .txt endenden Dateien im Arbeitsverzeichnis
- Stellen Sie die gelöschten Dateien mit einem einzigen Kommando mit Hilfe des Wildcard Zeichens ‚*‘ aus dem „backup“ Verzeichnis wieder her
- Wechseln Sie in das Verzeichnis „übung1“ und löschen Sie das Verzeichnis „backup“ samt Inhalt

Ein- und Ausgabe Umleitung

Daten Kanäle

- Ein „Daten Kanal“ stellt die Funktion für die Ein- und Ausgabe von Daten für einen Prozess zur Verfügung
- Interaktive Prozesse haben gewöhnlich 3 offene Kanäle
  - **Standard Input** (Kanal Nummer 0)
    - Für die Eingabe von Kommandos, Benutzer Daten, ...
    - Verbunden mit der Terminal Tastatur
  - **Standard Output** (Kanal Nummer 1)
    - Für die Ausgabe „normaler“ Meldungen und Daten
    - Verbunden mit dem Terminal Bildschirm
  - **Standard Error** (Kanal Nummer 2)
    - Für die Ausgabe von Sonder- und Fehlermeldungen
    - Verbunden mit dem Terminal Bildschirm
- Die Programmierung entscheidet, welcher Kanal wofür verwendet wird

Umleitung durch die Shell (1)

- Die Shell kann die Daten Kanäle vor der Ausführung eines Kommandos umleiten
  - Unabhängig vom jeweiligen Kommando und transparent für das Kommando
- Umleitung in eine beliebige Datei
  - `$ ls -l 1>/tmp/out.txt`
  - `$ cat country.txt >country.txt.backup`
- Umleitung auf ein beschreibbares Gerät
  - `$ echo Hallo >/dev/pts/1 # Pseudo Terminal`
  - `$ rm badfile  2>/dev/null # lässt Fehlermeldungen verschwinden`

Umleitung durch die Shell (2)

- Gleichzeitige Umleitung von Standard Output und Error
  - $ ls -l goodfile badfile >out.txt 2>err.txt # in zwei unterschiedliche Dateien
  - $ ls -l goodfile badfile >all.txt 2>&1 # in die selbe Datei
- Anhängen statt Überschreiben bei Umleitung
  - $ date >>out.txt
- Umleitung des Standard Inputs
  - $ echo y >answer # Speichern der Eingabe in der Datei answer
  - $ rm -i myfile <answer # Antwort aus Datei statt von der Terminal Tastatur

Übung: Daten Kanäle (1)

- Wechseln Sie in das Verzeichnis „übung1“
- Führen Sie das Kommando ls -l exercise baddir zum detaillierten Auflisten des existieren Verzeichnisses „exercise“ und des nicht existierenden Verzeichnisses „baddir“ aus
- Leiten Sie den Standard Output dieses Kommandos in die Datei „ls.out“ um, zeigen Sie danach den Inhalt der Datei an
- Leiten Sie den Standard Error dieses Kommandos in die Datei „ls.error“ um, zeigen Sie danach den Inhalt der Datei an
- Leiten Sie den Standard Output und Standard Error dieses Kommandos in die Datei „ls.all“ um, zeigen Sie danach den Inhalt der Datei an

Übung: Daten Kanäle (2)

- Verwenden Sie das echo Kommando, um mittels der Standard Output Umleitung die Zeile „Zeile 1 Datei1“ in die Datei „datei1.txt“ zu schreiben
- Hängen Sie an diese Datei die Zeile „Zeile 2 Datei1“ an, zeigen Sie danach den Inhalt der Datei an
- Erzeugen nach dem selben Schema die Datei „datei2.txt“
- Erstellen Sie mittels des cat Kommandos eine Datei „datei1+2.txt“, die aus dem Inhalt von „datei1.txt“ gefolgt von dem Inhalt von „datei2.txt“ besteht

Weitere Kommandos, Pipelines

Weitere Kommandos (1)

- Auflisten der laufenden Prozesse
  - `$ ps` # Prozesse der aktuellen Session
  - `$ ps -e` # alle („every“) Prozesse
  - `$ ps -f` # „full-format“ Listing der Session
  - `$ ps –u root` # alle Prozesse die dem Benutzer „root“ zugeordnet sind
- Beendigung eines Prozesses
  - `$ kill 1234` # Normales Beenden des Prozesse mit Process ID 1234
  - `$ kill -KILL 5678` # Unbedingter Abbruch des Prozesses mit Process ID 5678

Weitere Kommandos (2)

- Ausgeben aller Zeilen einer Textdatei, die eine Zeichenkette beinhalten
  - `$ grep Brutus friends.txt # alle Zeilen die die Zeichenkette „Brutus“ beinhalten`
- Anzeiger der Zeichen, Wörter und Zeilen einer Datei
  - `$ wc -l friends.txt # Zeilen, „lines“`
  - `$ wc -w friends.txt # Wörter`
  - `$ wc -c friends.txt # Zeichen, „character“`
- Auffinden von Dateisystemobjekten in einem Teilbaum des Dateisystems
  - `$ find /usr/share -name *fonts* # findet Objekte in /usr/share deren Namen „fonts“ beinhaltet`
  - `$ find /usr/share -name *fonts* -type d # ... und Directories sind`

Weitere Kommandos (3)

- Anzeigen der IP Adressen der Netzwerk Interfaces
  - `$ ip addr show`
- Anzeigen der IP Routing Table
  - `$ ip route show`
- Anzeigen der ARP Table
  - `$ ip neighbour show`

Pipelines (1)

- Eine Pipeline erlaubt die Weiterverarbeitung der Ausgabe eines Kommandos durch ein weiteres Kommando
  - Erlaubt die Generierung neuer Ergebnisse durch die Kombination mehrerer Kommandos
- Kommando Syntax: *cmd1* | *cmd2*
  - Paralleler Start der Kommandos *cmd1* und *cmd2* durch die Shell:
  - Standard Output von *cmd1* wird mit Standard Input von *cmd2* verbunden
  - Sind nicht auf zwei Kommandos beschränkt, können beliebig erweitert werden
- Es sind grundsätzlich alle Kommandos geeignet, die die Eingaben vom Standard Input lesen und/oder Ergebnisse am Standard Output ausgeben, z.B.:
  - more, wc, grep, sort, head, tail, ...

Pipelines (2)

- Seitenweise Anzeige eines langen, detaillierten Directory Listings
  - `$ ls -l /usr/share | more`
- Anzahl der angemeldeten Benutzer
  - `$ who | wc -l`
- Sortierte Liste der angemeldeten Benutzer
  - `$ who | sort`
- Ausabge der laufenden (bash) Shell Prozesse
  - `$ ps -e | grep bash`
- Anzahl der laufenden (bash) Shell Prozesse
  - `$ ps -e | grep bash | wc -l`

Übung: Pipelines

- Verwenden Sie die Kommandos `grep` und `wc` um herauszufinden, in wie vielen Zeilen das Wort „dagger“ in der Datei „dagger.txt“ vorkommt
- Verwenden Sie die Kommandos `ps` und `grep` um herauszufinden, welche laufenden Prozesse mit Ihrem Benutzernamen assoziiert sind
- Verwenden Sie die Kommandos `ps`, `grep` und `wc` um herauszufinden, wie viele Kommandos eine Verbindung zu Ihrem Benutzernamen haben
- Extrahieren Sie aus der ARP Table jenen Eintrag, der die MAC Adresse des Default Gateways/Routers angibt

vi(m) Text Editor

Warum vi?

- Konfigurationsdateien auf Unix/Linux Systemen sind typischerweise Text Dateien
- Graphische Oberflächen und graphische Editoren sind auf Server Systemen nicht üblich
- Konfigurationsänderungen werden daher oft mit einem Text Editor auf der Kommandozeile durchgeführt
- vi ist einer der ältesten und weit verbreitetsten Text Editoren

vi Modi

- **Kommando Modus**
  - Zum Navigieren innerhalb der Datei, löschen, zu anderen Modi wechseln, ...
  - Jeder Tastendruck ist ein Kommando
- **Erweiterter Kommando Modus**
  - Für weitere Kommandos, z.B. Suchen und Ersetzten
  - Kommando wird mit der Eingabetaste abgeschlossen
- **Einfüge Modus**
  - Zum Verändern des Inhaltes der Datei

Wechsel zwischen den Modi

- Kommando Modus -> Erweiterten Kommando Modus
  - Drücken der `:` Taste
  - Eingabezeile wird geöffnet
- Erweiterter Kommando Modus -> Kommando Modus
  - `<ESC>` Taste bricht ab
  - `<Eingabe>` Taste führt das erweiterte Kommando aus
- Kommando Modus -> Einfüge Modus
  - Drücken der Tasten: `i`, `a`, `I`, `A`, `c`, `C`, `o`, `O`, ...
- Einfüge Modus -> Kommando Modus
  - `<ESC>` Taste beendet den Einfüge Modus

Starten und Beenden des vi

- Starten von der Kommando Zeile
  - `$ vi textfile` # Datei „textfile“ editieren
  - `$ view textfile` # Datei „textfile“ im Read-Only Modus öffnen
  - `$ vi` # ohne Datei Argument, Datei wird aus dem Editor heraus geöffnet
    `:e textfile`
- Schließen der Sitzung aus dem Kommando Modus
  - `:q` Ausstieg („quit“) ohne Speichern, falls nichts verändert wurde
  - `:q!` Ausstieg ohne Speichern, nicht gespeicherte Änderungen werden verworfen
  - `:w` Speichern ohne die Sitzung zu beenden
  - `:wq` Speichern und Beenden
  - `ZZ` Speichern und Beenden

Cursor Navigation

`h` oder `←` eine Position nach links
`l` oder `→` eine Position nach rechts
`j` oder `↓` eine Position nach unten
`k` oder `↑` eine Position nach oben

`<STRG>-d` halbes Fenster nach unten
`<STRG>-u` halbes Fenster nach oben
`<STRG>-f` ganzes Fenster vorwärts
`<STRG>-f` ganzes Fenster rückwärts

`w` Wort vorwärts
`b` Wort rückwärts
`e` Ende des aktuellen Wortes

`G` letzte Zeile
`1G` erste Zeile
`123G` Zeile 123

`0` oder `<POS1>` Anfang Zeile
`$` oder `<ENDE>` Ende Zeile

Einfüge Modus

- Cursor an die gewünschte Position bewegen
- Nach erfolgten Einfügen mit `<ESC>` in den Kommando Modus zurück kehren
- Einfüge Kommandos
  - `a` nach der aktuellen Cursor Position anhängen
  - `A` am Ende der Zeile einfügen
  - `i` vor der aktuellen Cursor Position einfügen
  - `I` am Anfang der Zeile einfügen
  - `o` nach der aktuellen Zeile
  - `O` vor der aktuellen Zeile

Löschen von Text

- Gelöschter Text wird in einen internen Puffer kopiert und kann mit `p` danach wieder eingefügt werden
  - `x` oder `<ENTF>` löscht das Zeichen unter dem Cursor
  - `X` löscht das Zeichen links vom Cursor
  - `dd` löscht die gesamte Zeile
  - `13dd` löscht 13 Zeilen
  - `d0` löscht von der Cursor Position bis zum Anfang der Zeile
  - `d3w` löscht von der Cursor Position 3 Wörter weiter
  - `dG` löscht von der aktuellen Zeile bis zum Ende der Datei

Kopieren

- Text kann in einen internen Zwischenpuffer kopiert werden und mit `p` danach wieder eingefügt werden
  - `yy` kopiert die gesamte Zeile
  - `5yy` kopiert 5 Zeilen
  - `y$` kopiert von der Cursor Position bis zum Ende der Zeile
  - `p` fügt nach oder unterhalb der Cursor Position ein
  - `P` fügt vor oder oberhalb der Cursor Position ein

Suchen und Ersetzen

- Suche wird mit `/` im Kommando Modus gestartet
- Das Suchkommando startet eine Eingabe Zeile die mit `<EINGABE>` beendet wird
  - `/mytext` sucht in Richtung Ende der Datei nach „mytext“
  - `n` sucht nach dem nächsten Vorkommen des Text
- Suchen und Ersetzen wird im erweiterten Kommando Modus durchgeführt
  - `:s/old/new` ersetzt erstes „old“ in der aktuellen Zeile durch „new“
  - `:s/old/new/g` ersetzt jedes „old“ in der aktuellen Zeile durch „new“
  - `:5,22s/old/new/g` ersetzt in den Zeilen 5-22
  - `:1,$s/old/new/gc` ersetzt in der ganzen Datei, Bestätigung bei jedem Ersetzen

Weitere nützliche Kommandos

- `.` wiederholt das letzte Editierkommando
- `~` vertauscht Groß- und Kleinbuchstaben
- `J` verbindet zwei Zeilen
- `u` macht das letzte Editierkommando rückgängig
- `:w myfile` speichert als Datei „myfile“
- `:e myfile` schließt die aktuelle Datei und editiert „myfile“
- `:r myfile` fügt den Inhalt von „myfile“ an der aktuellen Position ein
- `:r! ls -l /etc` fügt die Ausgabe des Kommandos `ls -l /etc` an der aktuellen Position ein

Übung: vi

- Starten Sie das Kommando `vimtutor` um den vi Editor genauer kennen zu lernen