Studium neben dem Beruf

FOM Hochschule für Oekonomie & Management
University of Applied Sciences

Betriebssysteme Praxis

SS 2011

Hans-Georg Eßer
Dipl.-Math., Dipl.-Inform.

Foliensatz B (09.04.2011)
Dateien und Verzeichnisse, Editor vi

Einführung in die Linux-Shell

Praxisteil

- Arbeiten mit der Shell
- Verzeichnisnavigation, -Listings
- Dateien kopieren, umbenennen, verschieben
- Verzeichnisse erstellen, löschen etc.
- Dateien öffnen
- Der Editor „vi“

Shell-Prompt (1)

- Shell zeigt durch **Prompt** an, dass sie bereit ist, einen Befehl entgegen zu nehmen
- Prompts können verschieden aussehen:
  - … $ _
    … > _ : Anwender-Prompt, nicht-privilegiert
  - … # _ : Root-Prompt, für den Administrator

Shell-Prompt (2)

- Vor dem `$`, `>`, `#` meist Hinweise auf Benutzer, Rechner, Arbeitsverzeichnis

```
[esser@macbookpro:BS-Praxis]$
root@quad:~#
```

  - `esser`, `root`: Benutzername; individuell
  - `macbookpro`, `quad`: Rechnername
  - `BS-Praxis`, `~`: Arbeitsverzeichnis, je nach Prompt-Einstellung auch in voller Länge (z. B. /home/esser/Daten/FOM/SS2011/BS-Praxis)
  - `~` = „Home-Verzeichnis“ des Benutzers

Befehlseingabe (1)

- Am Prompt Befehl eingeben und mit [Eingabe] abschicken
- Shell versucht, (in der Regel) erstes Wort als Kommandoname zu interpretieren:
  - Alias? (→ später)
  - Shell-interne Funktion? (→ später)
  - eingebautes Shell-Kommando? (z. B. `cd`)
  - externes Programm? (Suche in Pfad)

Befehlseingabe (2)

- Beispiel: Aktuelles **Arbeitsverzeichnis** anzeigen (`pwd` = **p**rint **w**orking **d**irectory)

```
[esser@quad:~]$ pwd
/home/esser
[esser@quad:~]$ _
```

- Nach Abarbeiten des Befehls (oft: mit einer „Antwort“) erscheint wieder der Prompt – Shell ist bereit für nächstes Kommando

Befehlseingabe (3)

- Mehrere Befehle auf einmal abschicken: mit Semikolon `;` voneinander trennen

```
[esser@quad:~]$ pwd; pwd
/home/esser
/home/esser
[esser@quad:~]$ _
```

Befehlseingabe (4)

- **Inhaltsverzeichnis** anzeigen: `ls` (**list**)
- bezieht sich immer auf das aktuelle Arbeits-verzeichnis (Alternative: Ort als Parameter angeben)

```
[esser@quad:~]$ ls
bahn-2011-02-22.pdf     bh-win-04-kret.pdf
buch_kap08.pdf          bv-anleitung.pdf
bz2.pdf
[esser@quad:~]$ ls /tmp
cvcd  kde-esser  ksocket-esser  orbit-esser
ssh-vrUNLb1418  virt_1111
[esser@quad:~]$ _
```

Befehlseingabe (5)

- Inhalt mit mehr Informationen: `ls -l`

```
[esser@quad:~]$ ls -l
-rw------- 1 esser users  29525 Feb 21  2011 bahn-2011-02-22.pdf
-rw-r--r-- 1 esser users 745520 Apr 10  2004 bh-win-04-kret.pdf
-rw-r--r-- 1 esser users 856657 Oct 21  2005 buch_kap08.pdf
-rw-r--r-- 1 esser esser 738570 Mar 17 20:29 bv-anleitung.pdf
-rw-r--r-- 1 esser users 123032 Sep 22  2003 bz2.pdf
[esser@quad:~]$ _
```

- Ausgabe enthält zusätzlich:
  - Zugriffsrechte (`-rw-r--r--` etc.) → später
  - Dateibesitzer und Gruppe (`esser`, `users`) → später
  - Größe und Datum/Zeit der letzten Änderung

Befehlseingabe (6)

- Leere Datei erzeugen (für Experimente): `touch`

```
[esser@quad:~]$ touch Testdatei
[esser@quad:~]$ ls -l Testdatei
-rw-r--r-- 1 esser esser 0 Apr  7 13:58 Testdatei
[esser@quad:~]$ _
```

- Datei hat Größe 0

Befehlseingabe (7)

- Fehlermeldungen: Unbekanntes Kommando

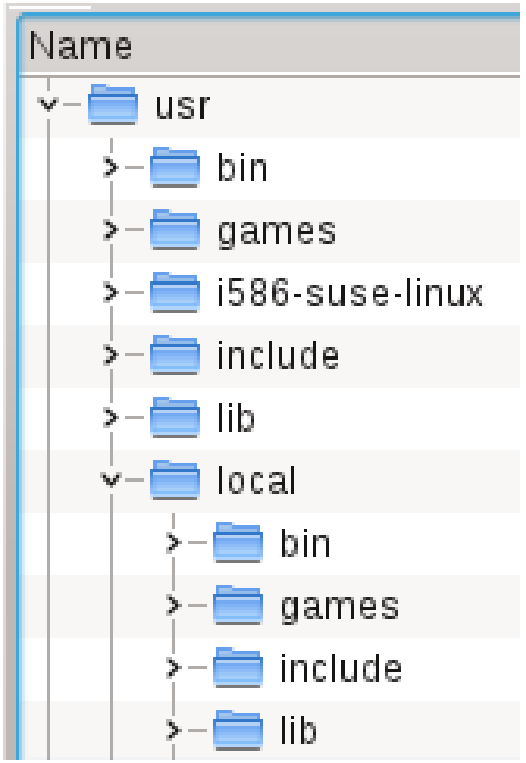
```
[esser@quad:~]$ fom
No command 'fom' found, did you mean:
 Command 'fim' from package 'fim' (universe)
 Command 'gom' from package 'gom' (universe)
 Command 'fop' from package 'fop' (universe)
 Command 'fdm' from package 'fdm' (universe)
 Command 'fpm' from package 'fpm2' (universe)
 [...]
fom: command not found
[esser@quad:~]$ _
```

- Meldung kann auch deutschsprachig sein

Dateiverwaltung (1)

Grundlagen (1)

- Linux kennt keine „Laufwerksbuchstaben“ (`C:`, `D:` etc.)
- Wurzelverzeichnis heißt `/`
- Pfadtrenner: auch `/` – d. h.: `/usr/local/bin` ist das Verzeichnis `bin` im Verzeichnis `local` im Verzeichnis `usr`. (wie bei Webadressen)



Dateiverwaltung (2)

Grundlagen (2)

- Weitere Datenträger erscheinen in Unterordnern
  - Beispiel: DVD mit Dateien zum Kurs hat Volume-Name `BS-ESSER`
  - Datei `test.txt` auf oberster DVD-Verzeichnisebene ist als `/media/BS-ESSER/test.txt` erreichbar (Windows: `e:\test.txt`)
  - Datei `Software/index.html` der DVD entsprechend als `/media/BS-ESSER/Software/index.html` (Windows: `e:\Software\index.html`)

Dateiverwaltung (3)

Grundlagen (3)

- Für private Nutzerdaten hat jeder Anwender ein eigenes **Home-Verzeichnis**, das i. d. R. unterhalb von `/home` liegt, z. B. `/home/esser`.
- Die Tilde `~` ist immer eine Abkürzung für das Home-Verzeichnis
  - funktioniert auch in zusammengesetzen Pfaden
  - `~/Daten/brief.txt` statt `/home/esser/Daten/brief.txt`

Dateiverwaltung (4)

Grundlagen (4)

- Ausnahme: Das Home-Verzeichnis des Systemadministrators `root` ist nicht `/home/root`, sondern `/root`
- Der Trick mit der Tilde `~` funktioniert aber auch für `root`
- Warum? `/home` könnte auf einer separaten Partition liegen und bei einem Fehlstart nicht verfügbar sein

Dateiverwaltung (5)

Grundlagen (5)

- Zwei Spezialverzeichnisse in jedem Ordner
  - `..` ist das Verzeichnis eine Ebene tiefer (von `/usr/local/bin` aus ist `..` also `/usr/local`)
  - `.` ist das aktuelle Verzeichnis
- Pfade kann man **absolut** und **relativ** zusammen bauen
  - absoluter Pfad beginnt mit `/`
  - relativer Pfad nicht; er gilt immer ab dem aktuellen Arbeitsverzeichnis

Dateiverwaltung (6)

Verzeichnisnavigation

- Kommando `cd` (**c**hange **d**irectory) wechselt in ein anderes Verzeichnis
- Zielverzeichnis als Argument von `cd` angeben – wahlweis mit relativem oder absolutem Pfad

```
[esser@quad:~]$ pwd
/home/esser
[esser@quad:~]$ cd /home ; pwd
/home
[esser@quad:home]$ cd .. ; pwd
/
[esser@quad:/]$ _
```

Dateiverwaltung (7)

Datei kopieren

- Kommando `cp` (**cop**y) kopiert eine Datei
- Reihenfolge: `cp` *`Original Kopie`*

```
[esser@quad:tmp]$ ls -l
-rw-r--r--  1 esser  wheel  1501 Apr  5 11:37 test.dat
[esser@quad:tmp]$ cp test.dat kopie.dat
[esser@quad:tmp]$ ls -l
-rw-r--r--  1 esser  wheel  1501 Apr  8 12:17 kopie.dat
-rw-r--r--  1 esser  wheel  1501 Apr  5 11:37 test.dat
[esser@quad:tmp]$ _
```

! Kopie erhält aktuelles Datum/Zeit

Dateiverwaltung (8)

Datei umbenennen

- Kommando `mv` (**mv**e) benennt eine Datei um
- Reihenfolge: `mv` *`AltName NeuName`*

```
[esser@quad:tmp]$ ls -l
-rw-r--r--  1 esser  wheel  1501 Apr  5 11:37 test.dat
[esser@quad:tmp]$ mv test.dat neu.dat
[esser@quad:tmp]$ ls -l
-rw-r--r--  1 esser  wheel  1501 Apr  5 11:37 neu.dat
[esser@quad:tmp]$ _
```

! Umbenennen ändert Datum/Zeit nicht

Dateiverwaltung (9)

Datei verschieben

- Kommando `mv` (move) verschiebt eine Datei
- Reihenfolge: `mv` *`AltName NeuerOrdner/`*

```
[esser@quad:tmp]$ ls -l
-rw-r--r--  1 esser  wheel  1501 Apr  5 11:37 test.dat
[esser@quad:tmp]$ mv test.dat /home/esser/
[esser@quad:tmp]$ ls -l
[esser@quad:tmp]$ ls -l /home/esser/
-rw-r--r--  1 esser  wheel  1501 Apr  5 11:37 test.dat
   [...]
[esser@quad:tmp]$ _
```

! Verschieben ändert Datum/Zeit nicht

Dateiverwaltung (10)

Datei löschen

- Kommando `rm` (**rem**ove) löscht eine Datei

```
[esser@quad:tmp]$ ls -l
-rw-r--r--  1 esser  wheel  1501 Apr  5 11:37 test.dat
[esser@quad:tmp]$ rm test.dat
[esser@quad:tmp]$ ls -l
[esser@quad:tmp]$ _
```

Dateiverwaltung (11)

Mehrere Dateien

- Einige Befehle akzeptieren mehrere Argumente, z. B.
  - `mv` (beim Verschieben in anderen Ordner)
  - `rm`
- Beispiele:

```
[esser@quad:tmp]$ mv datei1.txt datei2.txt Ordner/
[esser@quad:tmp]$ rm datei3.txt datei4.txt datei5.txt
[esser@quad:tmp]$ _
```

Dateiverwaltung (12)

Wildcards (*, ?)

- Bei Befehlen, die mehrere Argumente akzeptieren, können Sie auch Wildcards verwenden:
  - `*` steht für beliebig viele (auch 0) beliebige Zeichen
  - `?` steht für genau ein beliebiges Zeichen
- Beispiele:

```
[esser@quad:~]$ ls -l ?????.pdf
-rw-r--r--  1 esser  staff   79737 Apr  2 01:18 RegA4.pdf
-rw-r--r--  1 esser  staff  132246 Apr  4 18:02 paper.pdf
[esser@quad:~]$ rm /tmp/*
[esser@quad:~]$ _
```

Befehle testen

- Löschbefehl mit Wildcards zu gewagt?
  → vorher mit `echo` testen:

```
[esser@quad:Downloads]$ echo rm *.zip
rm Logo_a5_tif.zip Uebung1.zip c32dwenu.zip
ct.90.01.200-209.zip ct.90.12.130-141.zip
ct.91.02.285-293.zip ct.91.12.024-025-1.zip
ct.91.12.024-025.zip ct.92.08.052-061.zip
ix.94.03.010-011.zip ix.94.07.068-071.zip
[esser@quad:Downloads]$ rm *.zip
[esser@quad:Downloads]$ _
```

Wildcard-Auflösung

- Das letzte Beispiel verrät etwas über das Auflösen der Wildcards
  - Wenn Sie `rm *.zip` eingeben, startet die Shell *nicht* `rm` mit dem Argument „`*.zip`“
  - Die Shell sucht im aktuellen Verzeichnis alle passenden Dateien und macht jeden Dateinamen zu einem Argument für den `rm`-Aufruf.
  - Es wird also
    `rm Logo_a5_tif.zip Uebung1.zip c32dwenu.zip ct.90.01.200-209.zip ...`
    aufgerufen.

Verzeichnisse (1)

Mit Verzeichnissen können Sie ähnliche Dinge tun wie mit Dateien

- Verzeichnis erstellen
- (leeres!) Verzeichnis löschen
- Verzeichnis umbenennen oder verschieben
- Verzeichnis rekursiv (mit allen enthaltenen Dateien und Unterordnern) löschen

Verzeichnisse (2)

Verzeichnis erstellen

- Kommando `mkdir` (**make dir**ectory) erzeugt ein neues (leeres) Unterverzeichnis

```
[esser@quad:tmp]$ ls -l
[esser@quad:tmp]$ mkdir unter
[esser@quad:tmp]$ ls -l
drwxr-xr-x  2 esser  wheel  68 Apr  8 14:28 unter
[esser@quad:tmp]$ cd unter
[esser@quad:unter]$ ls -l
[esser@quad:unter]$ cd ..
[esser@quad:tmp]$ _
```

! Kurzform `md` für `mkdir` nicht immer vorhanden → vermeiden

Verzeichnisse (3)

Verzeichnis löschen

- Kommando `rmdir` (**rem**ove **dir**ectory) löscht ein leeres (!) Unterverzeichnis

```
[esser@quad:tmp]$ touch unter/datei
[esser@quad:tmp]$ rmdir unter
rmdir: unter: Verzeichnis nicht leer
[esser@quad:tmp]$ rm unter/datei
[esser@quad:tmp]$ rmdir unter
[esser@quad:tmp]$ _
```

! Kurzform `rd` für `rmdir` nicht immer vorhanden → vermeiden

Verzeichnisse (4)

Verzeichnis umbenennen / verschieben

- funktioniert wie das Umbenennen / Verschieben von Dateien
- gleicher Befehl: `mv`, wieder zwei Varianten:
  - `mv Verzeichnis NeuerName`
  - `mv Verzeichnis AndererOrdner/`

Verzeichnisse (5)

Verzeichnis rekursiv löschen

- Kommando `rm` (**rem**ove) hat eine Option `-r` zum **r**ekursiven Löschen:

```
[esser@quad:tmp]$ mkdir a; mkdir a/b; mkdir a/b/c
[esser@quad:tmp]$ touch a/b/c/datei
[esser@quad:tmp]$ rmdir a
rmdir: a: Verzeichnis nicht leer
[esser@quad:tmp]$ rm -r a
[esser@quad:tmp]$ _
```

! Vorsicht beim rekursiven Löschen: „Was weg ist, ist weg“

~~Undelete~~

- Undelete = Löschen rückgängig machen
  - gibt es unter Linux nicht
  - Wiederherstellung von gelöschten Dateien mit Profi-Tools möglich, wenn Computer nach dem Löschen sofort ausgeschaltet wurde
  - solche Tools stellen aber sehr viele Dateien wieder her → enormer Aufwand, anschließend die gesuchte Datei zu finden; u. a. sind die Dateinamen dauerhaft verloren

→ vor `rm -r` … mehrfach prüfen …

Optionen und Argumente

- **Argumente:** z. B. Dateinamen; beziehen sich oft auf Objekte, die manipuliert werden sollen
- **Optionen:** verändern das Verhalten eines Befehls
  - bei den meisten Befehlen zwei Varianten:
  - kurze Optionen: `-a`, `-b`, `-c`, ...
    → lassen sich kombinieren: `-abc` = `-a -b -c`
  - lange Optionen: `--ignore`, `--force`, `--all` etc.
  - Beispiel: `-r` bei `rm`

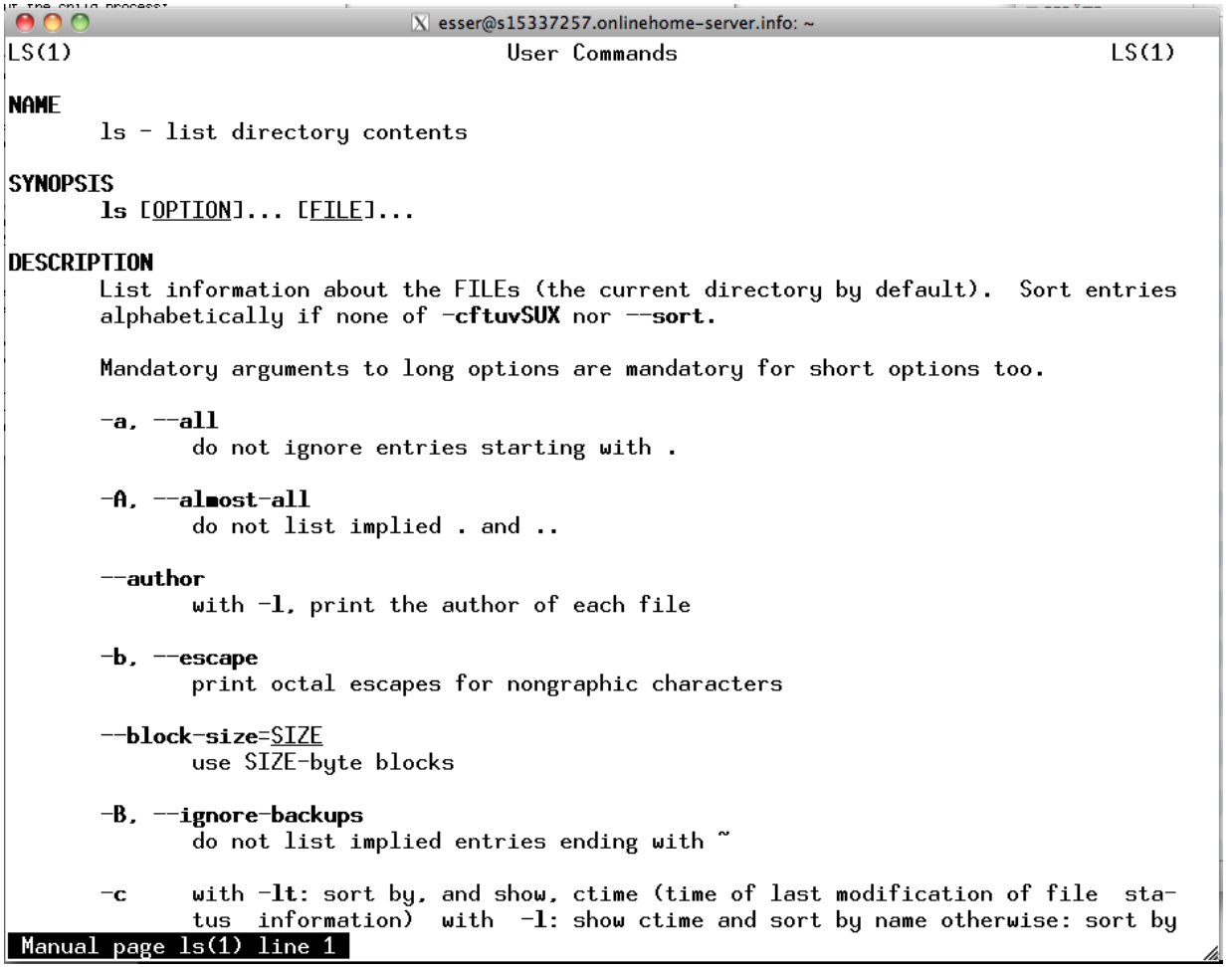
Hilfe: Handbuch

- Zu den meisten Kommandos gibt es eine sog. Manpage, die Sie über

  `man kommando`

  abrufen
- Beispiel:

  `man ls`



Der Editor vi (1)

- Standard-Editor auf allen Unix-Systemen (und damit auch Linux): `vi` (**vi**sual editor)
- gewöhnungsbedürftige Bedienung
- zwei Betriebsarten
  - **Befehlsmodus** (nach Start aktiviert; Normalmodus)
  - **Bearbeitungsmodus**
- `vi` aus Versehen gestartet? Verlassen ohne Speichern von Änderungen mit

  [Esc] `:q!`

Der Editor vi (2)

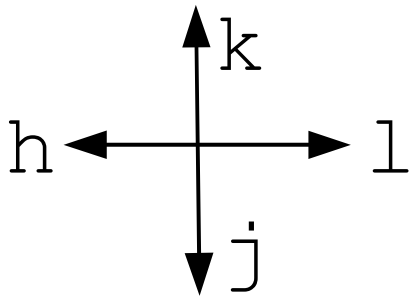
- Warum Umgang mit `vi` lernen?
  - auf jedem – noch so minimalistischen – Unix-System ist ein `vi` installiert (kleines Programm):

```
[esser@quad:~]$ ls -l /usr/bin/vi /usr/bin/emacs
-rwxr-xr-x 1 root root 5502096 Nov  9  2008 /usr/bin/emacs
-rwxr-xr-x 1 root root  630340 Oct 17  2008 /usr/bin/vi
```

  - läuft im Terminal → hilfreich bei Remote-Zugriff
  - Bei Problemen (Plattenfehler, nicht alle Datei-systeme verfügbar) sind andere Editoren evtl. nicht erreichbar, `vi` vielleicht doch → gilt leider nicht mehr für aktuelle Linux-Versionen
  - Thema ist LPI-prüfungsrelevant

Der Editor vi (3)

- Wechseln in den Bearbeitungsmodus: `i`, `I`, `a`, `A`
  - `i`: Text vor dem Cursor einfügen
  - `a`: Text nach dem Cursor einfügen
  - `I`: Text am Zeilenanfang einfügen
  - `A`: Text am Zeilenende einfügen
- Bearbeitungsmodus verlassen: [Esc]
- Navigieren im Text:
  Cursortasten oder:



Der Editor vi (4)

- Zeichen / Text löschen:
  - im Bearbeitungsmodus mit [Rückschritt] und [Entf], wie aus anderen Editoren bekannt
  - im Befehlsmodus mehrere Möglichkeiten:
  - `x` löscht Zeichen unter Cursor
  - `X` löscht Zeichen links von Cursor
  - `dw` löscht ab Cursor-Position bis Anfang des nächstens Wortes
  - `dd` löscht aktuelle Zeile
  - vorab Zahl: Mehrfachausführung (`15dd`: 15 Zeilen)

Der Editor vi (5)

- Speichern und beenden
  - Immer zuerst in den Befehlsmodus
    → im Zweifelsfall einmal [Esc] drücken
  - Speichern: `:w`
  - Speichern (erzwingen): `:w!`
  - Beenden (klappt nur, wenn Text seit letztem Speichern nicht verändert wurde): `:q`
  - Beenden erzwingen (ohne speichern): `:q!`
  - Speichern und beenden: `:wq` (oder: `ZZ` ohne „:“)

Der Editor vi (6)

- Suche im Text
  - Vorwärtssuche: `/` und Suchbegriff, dann [Eingabe]
  - Sprung zum nächsten Treffer: `n` (next)
  - Rückwärtssuche: `?` und Suchbegriff, dann [Eingabe]
  - Sprung zum nächsten Treffer: `n`
  - Wechsel zwischen Vorwärts- und Rückwärtssuche: einfach `/` bzw. `?` , dann Eingabe und mit `n` weiter (in neuer Richtung) suchen

Der Editor vi (7)

- Rückgängig machen / wiederherstellen
  - Letzte Änderung rückgängig machen: `u` (undo)
  - geht auch mehrfach: `u`, `u`, `u`, …
  - … und mit Mehrfachausführung: `3u` macht die letzten drei Änderungen rückgängig
  - Einen Undo-Schritt aufheben: [Strg]+`r` `:redo`
  - mehrfaches Redo: z. B. `3` [Strg]+`r`

Der Editor vi (8)

- Copy & Paste: Kopieren …
  - `yw` (ab Cursorposition bis Wortende)
  - `y$` (ab Cursorposition bis Zeilenende)
  - `yy` (ganze Zeile)
  - `3yy` (drei Zeilen ab der aktuellen)
- … und Einfügen
  - `P` (fügt Inhalt des Puffers an Cursorposition ein)
- Cut & Paste
  - Löschen mit `dd`, `dw` etc.; dann einfügen mit `P`

Der Editor vi (9)

- Copy & Paste mit der Maus
  - Wenn Sie die grafische Oberfläche verwenden, geht es auch mit der Maus:
  - Kopieren: Mauszeiger auf 1. Zeichen, klicken (und gedrückt halten), zum letzten Zeichen ziehen, loslassen
  - Einfügen: Cursor zu Ziel bewegen, dann (im Einfügemodus!) die mittlere Maustaste drücken
  - Bei beiden Schritten muss man je nach `vi`-Version evtl. die [Umschalt]-Taste drücken

Der Editor vi (10)

- Datei im Editor öffnen:

  ```
  [esser@quad:~] vi Dateiname
  ```

- zweite Datei an Cursorposition hinzuladen:

  ```
  :read Dateiname
  ```

  (im Befehlsmodus!)

Praktische Übungen

- Aufgabenblatt
  - Umgang mit Dateien und Verzeichnissen
  - Editor vi