Unix & Shell-Programmierung SS21

Vorlesungswoche 1

Helga Karafiat

FH Wedel

Infos zur Veranstaltung

Vorlesungsinhalte

- Fokus: Nutzung der Shell und Shell-Programmierung
- Allgemeine Informationen über Unix-Systeme und deren Aufbau und Funktionsweise
- ggfs. Infos und Tipps zur Anpassung und Konfiguration
- Nicht: wie installiere ich Linux, wie betreibe einen Server etc.

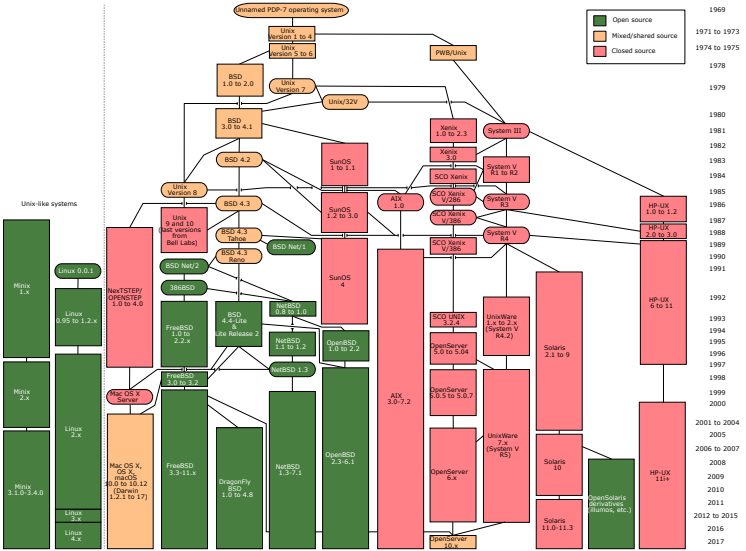
Material

- Unterlagen und Videos werden gemeinsam bei Moodle freigeschaltet
- “work in progress”

Zahlen, Daten, Fakten

- 1969 Ken Thompson (AT&T Bell Labs)
  erstes Unix-System auf PDP 7 von DEC, alles Gute aus Multics
- 1970 Ken Thompson and Dennis Ritchie (AT&T Bell Labs)
  Port von Unix auf PDP-11 minicomputer
- 1973 Systemkern von UNIX in C, Dennis Ritchie
- 1974 Quellcodelizenzen an Universitäten $\rightarrow$ z.B. Berkeley Unix (BSD)
- 1978 AT&T Version 7, hieraus viele UNIX Varianten abgeleitet
- 1979 AT&T gewährt keinen Einblick mehr in die Unix Quellen
- 1982 AT&T System III - erste offizielle Veröffentlichung
- 1983 AT&T System V - erstes Unix mit Support
- 1985 Richard Stallman gründet die FSF, schreibt GNU Manifesto
- 1989 Vereinheitlichung: System V Rel.4
- 1992 Linux, Linus Torvalds, freies UNIX mit GNU Software
- 2000 Mac OS 10, BSD basiert
- 2008 Android 1.0
- 2016 Windows Subsystem for Linux

Unix Zeittafel

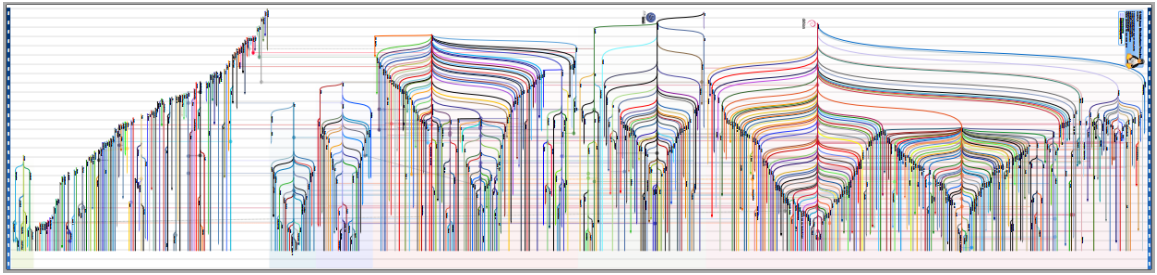


FSF, GNU und GPL

"GNU, which stands for Gnu's Not Unix, is the name for the complete Unix-compatible software system which I am writing so that I can give it away free to everyone who can use it."
(Richard Stallman, Gründer Free Software Foundation)

Linux

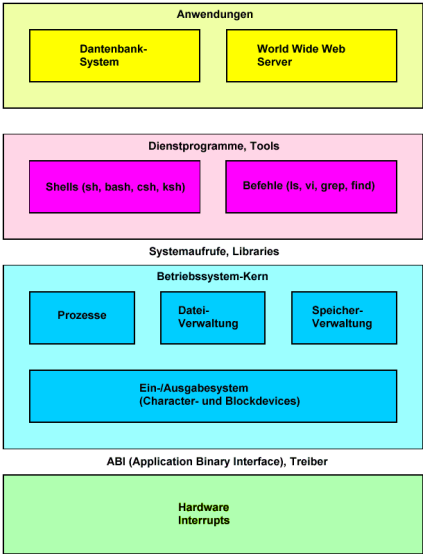
Die verschiedenen Linux-Distributionen mit Stammbaum ;)



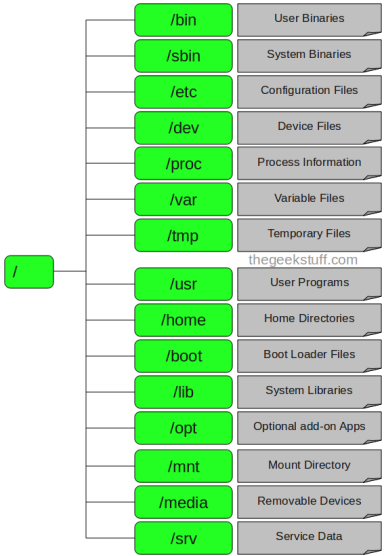
Unix Eigenschaften

- Unzählige Hardware-Plattformen: von Workstation über Hochleistungsserver bis hin zur Uhr
- Multiuser und Multitasking (schon immer)
- Gut vernetzbar (IP, TCP, UDP), Client-Server-Architektur
- Gute Unterstützung so ziemlich aller Programmiersprachen
- Grundsystem: nur Shell (Kommandozeile)
- Bildhafte Sprache ("The Elements Of Style: UNIX As Literature")
- Standardisierung durch POSIX (Portable Operating System Interface):
  - Kleinster gemeinsamer Nenner aller Unixe
  - Standardisierte C-API
  - Standardisierte Shell-Werkzeuge
- Grafische Oberfläche vom Grundsystem getrennt und optional
  - X-Server (z.B. X11, X.Org) kein fester Bestandteil des Betriebssystems
  - Netzwerkfähig
  - Viele Oberflächen - je nach Gusto (Xfce, KDE, GNOME, ...), auch mehrere parallel
- Hierarchisches Dateisystem mit genau einer Wurzel (kein c:, d:)
- sehr hohe Anpassbarkeit / Individualisierbarkeit

Aufbau des Systems



Dateisystem



Shell

- Benutzerschnittstelle und Interpreter (Kommandozeile)
- Interpretiert die Eingabe und führt die eingegebenen Kommandos aus
- Programmiersprache um Kommandos zu strukturieren und zu verbinden (shell skripting)
- Motivation: Einfache Werkzeuge zu komplexen Lösungen kombinieren
  - benötigt etwas Lernaufwand
  - schnell für komplexe Aufgaben!
  - hohe Automatisierung
- Viele verschieden Shells
  - sh (Bourne-Shell), dash (Debian-Almquist-Shell), bash (Bourne-Again-Shell), ksh (Korn-Shell), zsh (Z Shell)
  - csh, tcsh (C-Shells)
  - viele viele weitere

Verwendung der Tools / Befehle / Kommandos

- Syntax: Befehlsname + Optionen + Parameter (z.B. Dateiname)
- Optionen und Parameter von Befehl zu Befehl unterschiedlich
- Optionen beginnen üblicherweise mit:
  - `--` (lange "gesprächige" Version) oder
  - `-` (kurze Version, nicht für alle Optionen immer vorhanden)
- Einige Konventionen bei Optionen, wie z.B.
  - `--help`, `-h`: falls vorhanden Ausgabe der usage
  - `--version`: mindestens Ausgabe der Programmversion
  - `--verbose`: "geschwätzige" Programmausgabe
- Trennung durch Leerzeichen
  - Standardtrenner
  - Leerzeichen in Dateinamen sind böse!

- Schema: `CMD_NAME [OPTION]... [PARAMETER]...`
  - `[]` optionale Angabe (Angaben ohne `[]` müssen gemacht werden)
  - `...`: beliebig viele von dem davor
- Beispiele
  - `touch [OPTION]... FILE...`
    Keine bis mehrere Optionen und mindestens ein Dateiname
    z.B. `touch foo.txt`
  - `echo [OPTION]... [STRING]...`
    Keine bis mehrere Optionen und kein bis mehrere Eingabestrings möglich
    z.B. `echo -e Hallo Welt`

Navigation

- Ausgabe des akuellen Verzeichnisses: `pwd`
- Ausgabe der aktuellen Ordnerinhalte: `ls`
- Verzeichnis wechseln (absolute oder relative Pfade): `cd`
  - `cd .` bleibt im aktuellen Verzeichnis
  - `cd ..` ein Verzeichnis nach oben
  - `cd ~` oder `cd` wechselt ins home Verzeichnis
  - `cd -` wechselt ins zuletzt besuchte Verzeichnis
  - Tab-Completion nutzen! (Verzeichnisse/Pfade “tabben”, nicht abtippen ;))

Hilfe?!

- manpages: `man CMD-NAME`
  - ausführliche Erläuterungen zu allen Unix-Tools
  - eingeteilt in Sections je nach Art des Programms (1-8)
  - Handbuch mit weitestgehend standartisiertem Aufbau (Sections: Name, Synopsis, Description ...)
  - z.B. `man ls` oder `man man`
- helppages: `help CMD-NAME`
  - Hilfe zu builtin-Funktionen
  - Aufruf von `help` bietet Übersicht über alle buitlin-Funktionen
  - Hilfreiche Option: `-s` liefert nur die Syntax eines Befehls
  - z.B. `help cd`
- infopages: `info CMD-NAME`
  - gedacht als "modernere" Variante der manpages
  - manche Tools haben nur noch infopages

- Hilfeausgabe mit Option `--help` (oder `-h`)
  - häufig eine etwas kürzere Hilfe zur Verwendung (usage)
  - nicht immer konsequent umgesetzt
  - z.B. `ls --help`
- Weitere nützliche Dinge
  - `apropos`: durchsuchen der manpages nach Schlagworten,
    z.B. `apropos directory` oder `apropos directories`
  - `whatis`: einzeilige Funktionsbeschreibung aus der manpage
  - Tab-Completion (auch bei Befehlen reichen die ersten Buchstaben ;))
  - Bash-History (“Hatte ich das nicht letztens schon getippt?”)
    - Bereits eingegebene Befehle über Pfeiltasten (hoch, runter)
    - Suchen mit `Strg + R`

Dateiverwaltung - Teil 1

- Verzeichnis(se) erstellen / löschen:
  `mkdir NAME...`, `rmdir NAME...`
- Kopieren von Dateien:
  `cp NAME1 NAME2`
- Kopieren von Dateien in ein Verzeichnis:
  `cp NAME1... DIR`
- Verschieben von Dateien in ein Verzeichnis:
  `mv NAME1... DIR`
- Überschreiben eines Dateinamens:
  `mv NAME1 NAME2`
- Löschen von Dateien:
  `rm NAME...` (Vorsicht bei `rm -r` und `rm -rf`)

Wildcards - Suchmuster im Dateisystem (globbing)

- `*` beliebige Zeichenfolge
  - Beispiele: `ls *`, `ls *.*`, `ls *.txt`
- `?` ein beliebiges Zeichen
  - Beispiele: `ls ???`, `ls ???*`
- `[abc]` ein Zeichen aus der Zeichenmenge a,b,c
  - Beispiel: `ls [abc]*`
- `[0-9]` ein Zeichen aus dem Zeichenintervall 0,1,..,9
  - Beispiel: `ls [0-9].*`
- `[a-z,A-Z]` ein Zeichen aus dem Zeichenintervall a,b,..,z,A,B,..,Z
  - BeispieL `ls [a-h]*`
  - Vorsicht: Semantik abhängig von eingestellter Sprache:
    Bei z.B. Deutsch oder Englisch keine Unterscheidung von Groß- und Kleinbuchstaben
- `[!...]` bzw `[^...]` negiert was an Stelle der ... angegeben ist
  - Beispiele: `ls [!a]*`, `ls [!abc]*`, `ls [!a-g]*`, `ls [^a]*`

Dateiattribute

- Anzeige mit `ls -l`
- Dateiarten (erstes Zeichen im "Attributblock")
  - `-` einfache Dateien, files
  - `d` Verzeichnisse, directories
  - `c` / `b` zeichen- / blockorientierte Gerätedateien, character/block devices
  - `l` symbolische Verweise, symbolic links, soft links
  - `p` benannte Röhren, named pipes
  - `s` Steckdosen, sockets
- Dateirechte (restlichen Zeichen im "Attributblock")
  - 3 Klassen: user, group, others
    - 3 Attribute pro Klasse: read (r), write (w), execute (x)
  - Bei Ordnern:
    - r: Verzeichnis lesen: nur Dateinamen
    - w: Dateien im Verzeichnis erzeugen und löschen
    - x: Suchen im Verzeichnis: Dateiattribute lesen

- Weitere Angaben
  - Anzahl der Hardlinks
  - Besitzer und Gruppe der Datei
  - Dateigröße (`-h` für human readable)
- verschiedene Zeitstempel
  - letztes Lesen: `ls -l -u`
  - letztes Schreiben: `ls -l -c`
  - letztes Modifizieren der Attribute: `ls -l -i`
  - Alle Informationen für eine Datei mit `stat FILE...`
- Versteckte Dateien: beginnen mit `.` (Anzeige mit `ls -a`)

- Ändern der Dateiattribute
  - Zugriffsrechte: `chmod`
    - Numerischer Modus:
      `chmod OCTAL-MODE FILE...`
      Oktalzahlen (0-7) bestimmen welche Flags gesetzt werden (3 Bit in Binärdarstellung)
      Beispiel: `chmod 754 beispiel.sh`
    - Symbolischer Modus:
      `chmod MODE FILE...`
      MODE: [ugoa][+-=][rwx]
      Vorteil: manchmal quick and dirty ;), z.B. `chmod +x beispiel.sh`
      Nachteil: wird schnell unübersichtlich, z.B. `chmod u=rwx,g=rx,o=r beispiel.sh`
  - Besitzer / Gruppe: `chown`
  - Gruppe: `chgrp`
  - Zeitstempel: `touch`

Dateiverwaltung - Teil 2 (Dateiinhalte ausgeben)

- `cat [FILE]...`
  gibt den Inhalt von `FILE` auf der Konsole aus
- `head [FILE]...` / `tail [FILE]...`
  gibt die ersten / letzten Zeilen von `FILE` auf der Konsole aus
- `more [FILE]...` / `less [FILE]...`
  gibt `FILE` seitenweise aus (plus diverse nette Features wie Suche etc.)

Weitere nützliche Tools

- `grep PATTERN [FILE]...`
  sucht in `FILE` nach Vorkommen des angegeben Wortes `PATTERN`
- `sort [FILE]...`
  sortiert die Zeilen von `FILE`
- `wc [FILE]...`
  zählt die Zeilen, Worte und Zeichen von `FILE`

Kommandozeilen-Editoren

- Vorteile
  - überall verfügbar
  - keine GUI vonnöten
  - Bedienung nur über die Tastatur (auch für Sonderfunktionen)
  - nach Einarbeitungszeit deutlich schneller als “mit der Maus klicken”
  - Konsole muss nicht verlassen werden
- Nachteile (sofern man davon sprechen kann)
  - Einarbeitungszeit
  - Lernkurve
- Beispiele
  - `nano` (ein wenig Windows-User-freundlich)
  - `emacs` (Kommandozeileneditor und Betriebssystem)
  - `vi / vim` (the one and only :))

VI / VIM

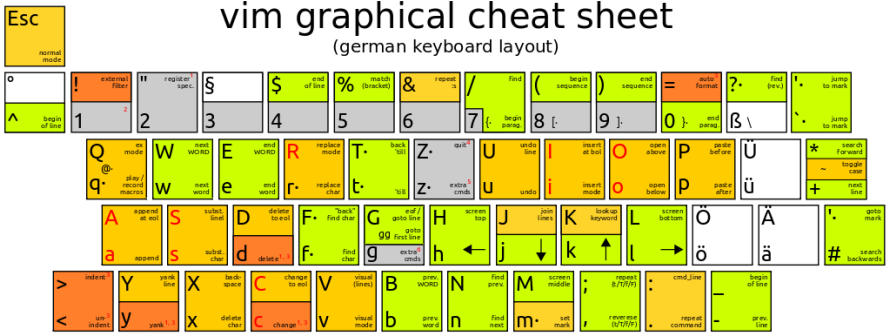
- sehr verbreitet, auf so gut wie jedem Unix vorhanden
- schnell, speicherplatzschonend, nach Einarbeitungszeit extrem effizient zu bedienen
- sehr gutes Syntaxhighlighting (auch für Shell)
- Verschiedene Modi (Kurzübersicht):
  - normal mode [ESC]
    - Navigieren in der Datei
      `h`, `j`, `k`, `l` (links, unten, oben, rechts)
      `Strg + f` / `Strg + b` eine Bildschirmseite weiter / zurück
      Nur vim: Navigation mit Pfeiltasten
    - Suchen mit `/SUCHWORT`
    - Kopieren, Ausschneiden, Einfügen etc.
      `y`: Kopieren (`yy`: ganze Zeile, `yw`: Cursor bis Wortende, ...)
      `x`: Einzelnes Zeichen ausschneiden
      `d`: Ausschneiden (`dd`: ganze Zeile, `dw`: Cursor bis Wortende, ...)
      `p`: Einfügen (`p`: nach aktueller Zeile / Cursor, `Shift + p`: vor aktueller Zeile / Cursor)
    - Wechsel in andere Modi

- Verschiedene Modi (Kurzübersicht - Fortsetzung):
  - command-line mode (ex mode) [:]
    - ★ Speichern, Öffnen, etc.
      `e FILE`: Datei FILE öffnen
      `split FILE` / `vsplit FILE`: geteiltes Fenster mit Datei FILE und aktuell offener Datei
      Fenster wechseln: `Strg+w Richtung`, `Strg+w w`
      `w [FILE]`: Datei speichern bzw. Kopie unter dem Namen FILE anlegen
      `q`: Editor verlassen, häufig in Kombination mit Speichern: `wq`
      Suffix `!`: "Ist mir egal, mach es!", z.B. `q!` Änderungen verwerfen und beenden
    - ★ Setzen von Einstellungen für die aktuelle Session
      z.B: `syntax on`, `set enc=utf-8`, `set tabstop=ZAHL`, `set expandtab`, `set number`, ...
    - ★ Weitere nützliche Befehle
      z.B. `help`, `retab`, `smi`, ...
    - ★ andere Shellbefehle ausführen mit Präfix `!`
      z.B. `!ls`, `!ls -l`, ...

- Verschiedene Modi (Kurzübersicht - Fortsetzung):
  - insert mode [`i`, `a`, `o`, `I`, `A`, `O`]
    - Texte schreiben
    - Nur vim: Navigation mit Pfeiltasten
  - visual modes: visual / visual block / visual line [`v` / `Ctrl+v` / `Shift+v`]
    - Markieren
    - Arbeiten auf Auswahl (analog zum normal mode)
- Persistente Einstellungen pro User `~/.vimrc`
- `vimtutor`: Lernen von vim anhand einer Datei

vim graphical cheat sheet

(german keyboard layout)



motion	moves the cursor or defines the range for an operator
command	direct action cmd, if red it enters insert mode
operator	requires a motion afterwards, operates between cursor and destination
extra	special functions, requires extra input
q·	commands with a dot need a char argument afterwards

Main command line commands (":ex"):

:w [file] (save)
:q (quit)
:q! (quit without saving)
:wq (save & quit)
:e foo (open file foo)
:n (new file)
:sp (split window horizontally)
:vsp (split window vertically)
:reg (display content of named registers)
:Explore [dir] (open file-explorer)
:h (help)
:h holy-grail (list all commands)

Other important commands:

CTRL + R (redo)
CTRL + P / N (complete the current word)
CTRL + W (move cursor to next window)
[n] CTRL + 6 (toggle [n]th alternate file)
CTRL + F / B (page up / down)
CTRL + E / Y (scroll lin eup / down)
CTRL + V (block-visual mode)

Find and replace:

:%s/<RegExp>/<String>/g (replace <RegExp> by <String> filewide)
:s/<RegExp>/<String>/ (search current line and replace first match)

Vim 7.x only commands:

CTRL + X - CTRL + O (omni completetion in insert mode)

:tabe [file] (edit [file] in a new tab)
:tabc [n] (close tab [n])
:tabonly (close all other tab pages)
:tabmove [n] (move tab to position [n])
:tab [cmd] (execute [cmd] and when it opens a new window open a new tab page instead, e.g :tab split opens current buffer in a new tabm :tab help gt opens tab page with help for gt)
:tabs (list all tab pages)
[n] gt (goto next or tab [n])
gT (goto previous tab)

:undolist (list leafs in tree of change)
:earlier [n] [s/m/h] (goto older text state [n] times / sec / min / hours)
g- (goto older text state)
:later [n] [s/m/h] (goto newer txt state [n] times / sec / min / hours)
g+ (goto newer text state)

Notes:

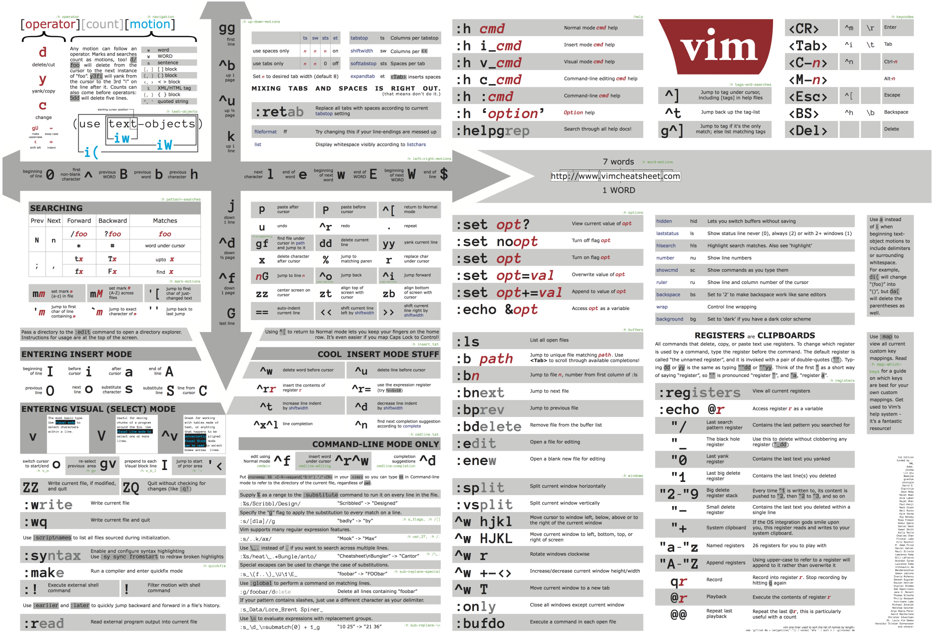
[1] use "x before a yank/paste/del command to use that register ('clipboard') (e=a..x,*)
(e.g.:" ay5 to copy rest of line to reg 'a')
[2] type in a number before any action to repeate it that number of times
(e.g.: 2p, d2w, 5i, d4j)
[3] duplicate operator to act on current line
(dd = delete line, >> = intent line)
[4] ZZ to save & quit, ZQ to quit w/o saving
[5] zt: scroll cursor to top, zb bottom, zz: center
[6] gg: top of file (vim only), gf: open file under cursor (vim only)

Visual mode:

Move around and type operator to act on the selected region.

Vim-Help navigation:

CTRL + ALT GR +] or :ta [:tag] (jump to subject using tags, CTRL + O to jump back)


[operator][count][motion]
d delete/cut
y yank/copy
c change
Any motion can follow an operator. Marks and searches count as motions, too! d/foo will delete from the cursor to the next instance of "foo". y3fi will yank from the cursor to the 3rd "i" on the line after it. Counts can also come before operators: 5dd will delete five lines.
(use text-objects)
gg first line
^b up 1 page
^u up ½ page
k up 1 line
j down 1 line
^d down ½ page
^f down 1 page
G last line
MIXING TABS AND SPACES IS RIGHT OUT.
:retab Replace all tabs with spaces according to current tabstop setting
Try changing this if your line-endings are messed up
Display whitespace visibly according to listchars
beginning of line 0
first non-blank character ^
previous WORD B
previous word b
previous character h
next character l
end of word e
beginning of next word w
end of WORD E
beginning of next WORD W
end of line $
SEARCHING
Prev Next Forward Backward Matches
ENTERING INSERT MODE
ENTERING VISUAL (SELECT) MODE
ZZ Write current file, if modified, and quit
ZQ Quit without checking for changes (like :q!)
:write Write current file
:wq Write current file and quit
Use :scriptnames to list all files sourced during initialization.
:syntax Enable and configure syntax highlighting
:make Run a compiler and enter quickfix mode
:! Execute external shell command
! Filter motion with shell command
Use :earlier and :later to quickly jump backward and forward in a file's history.
:read Read external program output into current file
COOL INSERT MODE STUFF
COMMAND-LINE MODE ONLY
:h cmd Normal mode cmd help
:h i_cmd Insert mode cmd help
:h v_cmd Visual mode cmd help
:h c_cmd Command-line editing cmd help
:h :cmd Command-line cmd help
:h 'option' Option help
:helpgrep Search through all help docs!
vim
^] Jump to tag under cursor, including [tags] in help files
^t Jump back up the tag-list
g^] Jump to tag if it's the only match; else list matching tags
<CR> ^m \r Enter
<Tab> ^i \t Tab
<C-n> ^n Ctrl-n
<M-n> Alt-n
<Esc> ^[Escape
<BS> ^h \b Backspace
<Del> Delete
7 words
http://www.vimcheatsheet.com
1 WORD
:set opt? View current value of opt
:set noopt Turn off flag opt
:set opt Turn on flag opt
:set opt=val Overwrite value of opt
:set opt+=val Append to value of opt
:echo &opt Access opt as a variable
:ls List all open files
:b path Jump to unique file matching path.
:bn Jump to file n, number from first column of :ls
:bnext Jump to next file
:bprev Jump to previous file
:bdelete Remove file from the buffer list
:edit Open a file for editing
:enew Open a blank new file for editing
:split Split current window horizontally
:vsplit Split current window vertically
^w hjkl Move cursor to window left, below, above or to the right of the current window
^w HJKL Move current window to left, bottom, top, or right of screen
^w r Rotate windows clockwise
^w +-<> Increase/decrease current window height/width
^w T Move current window to a new tab
:only Close all windows except current window
:bufdo Execute a command in each open file
REGISTERS are CLIPBOARDS
:registers View all current registers
:echo @r Access register r as a variable
qr Record
@r Playback
@@ Repeat last playback

Dateiströme

- Alle Programme schreiben auf die Standardausgabe, viele lesen von der Standardeingabe
- Standardausgabe (`stdout`)
  - Ausgabestrom, der der Ausgabe von Daten aus einem Programm dient
  - Standardmäßig mit der Konsole verbunden (unter Unix `/dev/stdout`, Deskriptor 1)
  - Umleitung über `1>` und `1>>`
    - `CMD 1> FILE` schreibt die Ausgabe von CMD in FILE, Inhalte werden überschrieben
    - `CMD 1>> FILE` hängt die Ausgabe von CMD am Ende von FILE an
    - Kurzschreibweise (wird üblicherweise verwendet):
      `CMD > FILE` bzw. `CMD >> FILE`
    - Noch nicht vorhandene Dateien werden bei beiden Umleitungen erzeugt
    - Beispiel: `echo Hallo > hallo.txt`

- Standardeingabe (`stdin`)
  - Eingabestrom, der der Eingabe von Daten in ein Programm dient
  - Standardmäßig mit der Tastatur verbunden (unter Unix `/dev/stdin`, Deskriptor 0). Klassisch: "blinkender Cursor".
  - Beenden der Eingabe mit *STRG+D* (sendet EOF)
  - Umleitung über `0<`
    - `CMD 0< FILE` schreibt den Inhalt von FILE auf die Eingabe von CMD
    - Kurzform (übliche Schreibweise): `<`
    - Beispiel: `tr -d a < beispiel.txt`
- Fehlerausgabe (`stderr`)
  - Weiterer Ausgabestrom speziell für Fehler- und Statusmeldungen (unter Unix `/dev/stderr`, Deskriptor 2)
  - Standardmäßig Mischen von `stdout` und `stderr`
  - Umleitung über `2>`
  - Beispiel: `ls nichtda 2> fehler.txt`
  - Viele Tools bieten Optionen an um die Fehlerausgabe zu unterdrücken (z.B. `--quiet`)

Dateiverwaltung - Teil 3 (Dateien erstellen)

- Datei mit Inhalt erzeugen durch Ausgabeumleitung
  - Beispiel: `echo Hallo Welt > foo.txt`
  - Wenn Datei schon vorhanden wird Inhalt überschrieben
- Inhalt an Datei anhängen durch Ausgabeumleitung
  - Beispiel: `echo Noch ein Hallo Welt >> foo.txt`
- Leere Datei erzeugen ("durch die Hintertür") mit `touch NAME1...`
  - Beispiel: `touch foo.txt`
  - Wenn Datei schon vorhanden nur Änderung des Zeitstempels
- Brace Expansion: mehrere Dateien / Verzeichnisse nach bestimmtem Schema erstellen
  - Vorsicht: `bash`-Erweiterung, nicht POSIX-konform!
  - Erzeugt Namen durch Expansion ("Permutation")
  - Syntax: `{abc,def,ghi}` für Listen oder `{a..h}` für Bereiche
  - Beispiel: `touch bar{0..3}.{txt,sh}`

Das erste Skript

- Motivation
  - Automatisierung von wiederkehrenden Aufgaben
  - weniger Tipparbeit auf der Konsole
  - Persistenz und Portabilität
  - Erweiterbarkeit
- Voraussetzungen
  - Angabe des Interpreters mit Shebang und vollständigem Pfad in physikalisch erster Zeile: `#!/bin/sh`
  - Ein Befehl pro Zeile, Befehle werden nacheinander ausgeführt
  - Kommentare mit # (bis zum Ende der Zeile)
  - Skript muss ausführbar gemacht werden (`chmod`)
  - Ausführung mit konkreter Angabe des Pfades `./SKRIPT-NAME`
- Sinnvolles Vorgehen
  - Erster Schritt: Ausprobieren der Befehle auf der Konsole
  - Zweiter Schritt: Einfügen der Befehle ins Skript

- Beispiel: TODOs in Dateien im aktuellen Verzeichnis finden

listtodo.sh (Erste Version)

```
#!/bin/sh
# Sucht nach TODOs in allen Dateien im Verzeichnis

grep TODO *
```