Vi für Fortgeschrittene

Vortrag im Rahmen des Proseminars Unix-Tools
am 3.5.2005

Jonathan Kleinehellefort <jk@molb.org>

Inhaltsverzeichnis

Terminal-Grundlagen

Als Terminal bezeichnet man ein **Ein-/Ausgabegerät**, dass über ein Kabel oder Netzwerk mit einem Computer verbunden ist.

Geschichte

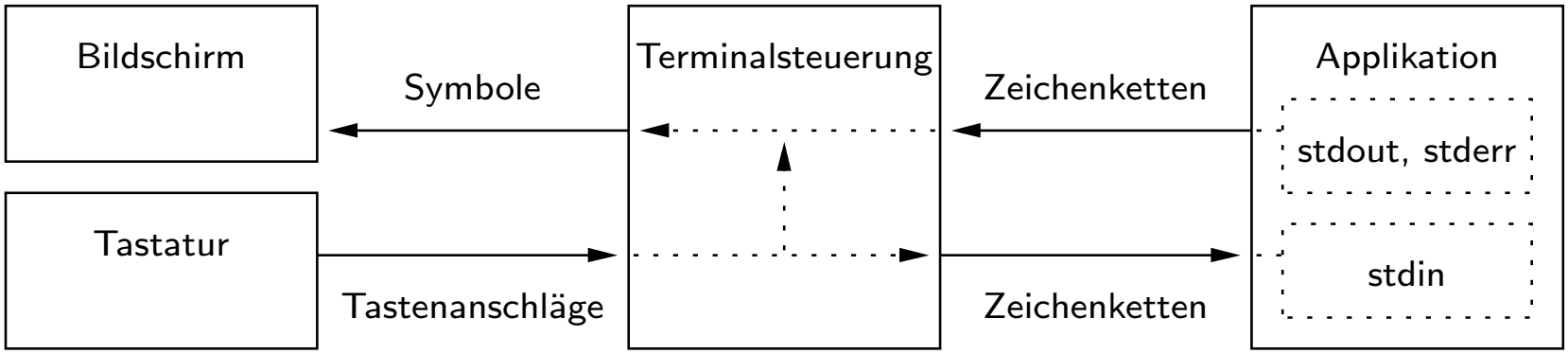
Die ersten Terminals waren so genannte zeichenbasierte **Teletypes** (**TTYs**), die aus einem **Drucker** (später **Monitor**) und einer **Tastatur** bestanden.

Da viele Unix-Programme zur Ein- und Ausgabe ein Terminal erwarten, gibt es auf heutigen Unix-Systemen **Terminal-Emulatoren**.

Funktionsweise

Tastatureingaben werden an die **Standardeingabe** der darin laufenden Applikation gesendet. Die **Standardausgabe** sieht man auf dem Schirm.

Einige Ein- und Ausgaben werden vom Terminal selbst speziell interpretiert.



Kontroll-Zeichen

Kontroll-Zeichen sind **nicht-druckbare** ASCII-Zeichen (Werte 0–31).

In Vi werden sie folgendermaßen dargestellt: `^M` (Newline), `^[` (Escape).

Kontroll-Zeichen werden durch bestimmte Tasten erzeugt, oder indem man einen Buchstaben zusammen mit dem **Control-Modifier** eintippt.

Eingabe in Vi

Manchmal ist es notwendig, ein Steuerzeichen in Vi einzugeben. Damit dieses als Eingabe und **nicht als Befehl** interpretiert wird, muss man es **„escapen“**, indem man ihm ein `^V` voranstellt.

Escape-Sequenzen

Um spezielle **Steuerbefehle** an ein Terminal zu senden, werden so genannte Escape-Sequenzen verwendet.

Escape-Sequenzen werden auch beim betätigen **mancher Tasten** vom Terminal an die Anwendung gesendet.

Escape-Sequenzen beginnen mit einem Escape-Zeichen (`^[`).

Beispiele

- Linke Cursor-Taste: `^[[D`
- Bildschirm löschen: `^[[2J`

Nutzung externer Programme

Shell-Befehle

`:!<shcmd>^M`

Dieses Kommando führt den Befehl *<shcmd>* in einer Shell aus.

Dies ist vor allem nützlich, um Dateioperationen zu tätigen, ohne den Editor verlassen zu müssen.

Beispiele

```
:!chmod 755 script^M
:!ls -l^M
:!./script^M
```

Ausführen von Make

In **Vim** gibt es extra einen Befehl, um `make` aufzurufen:

`:make[ <target>]^M`

Mit Hilfe von `:cc`, `:cn` und `:cp` kann man sich dann einen Fehler anzeigen lassen und zum nächsten oder vorherigen springen.

Das Ausgabeformat lässt sich je nach Compiler einstellen.

Filter

Es lassen sich ausserdem **Zeilen** des Editorpuffers durch Shell-Befehle filtern, d.h. einige Zeilen werden auf die **Standardeingabe** eines Programmes gegeben und durch dessen **Ausgabe** ersetzt:

`!<motion><shcmd>^M`

<motion> ist dabei ein **Bewegungskommando** und *<shcmd>* der **Filter-Befehl**.

Beispiele

- Einen Absatz „formatieren“: `{!}fmt^M`
- Alle Zeilen alphabetisch sortieren: `gg!Gsort^M`

Makros

Ein Makro ist eine **Befehlssequenz** (ein Skript), die einen bestimmten Ablauf in einer Anwendung **automatisiert**.

Abkürzungen

`:ab <lhs> <rhs>^M`

Danach wird jede Eingabe von *<lhs>* im **Eingabemodus** durch *<rhs>* ersetzt, solange *<lhs>* nicht Teil eines Wortes ist.

<rhs> kann auch **Steuerzeichen** (insbesondere auch Escape) enthalten.

Remapping von Tastatureingaben

`:map` *<lhs>* *<rhs>*`^M`

Tippt man nun im **Befehlsmodus** *<lhs>*, wird stattdessen *<rhs>* ausgeführt.

Für **Eingabemodus**-Mappings benutzt man `:map!`.

Achtung: Kontrollzeichen müssen mit `^V` maskiert werden!

Beispiele

Definieren einer LaTeX-Typewriter-Schrift:

```
:map! TTT \texttt{}^V^[i^M
:map TTT lbi\texttt{^V^[ea}^V^[^M
```

Textpuffer-Macros

Benutzung von Textpuffern

Man kann den Puffer (**Register**), den ein **Kopier- oder Einfügebefehl** benutzt, bestimmen, indem man ihm ein `"<letter>` voranstellt.

Beispiele

- Ein Wort im Puffer **a** speichern: `"ayw`
- Das Wort hinter dem Cursor einfügen: `"ap`

Eingaben „aufnehmen“ und „abspielen“

Eine weitere Möglichkeit, in einen solchen Puffer zu schreiben, ist, die Eingaben in ihm „aufzunehmen“ (nur in Vim):

`q<letter><seq>q`

Mit `@<letter>` lassen sie sich dann ausführen.

Beispiele

- **Aufnehmen:** Löschen des letzten Zeichens der Zeile: `qq$xjq`
- **Ausführen:** Für die nächsten 20 Zeilen wiederholen: `20@q`

Einstellungen

Anzeigen

Eine Liste aller Einstellungen kann man sich mit `:set all^M` anzeigen lassen.

Den Wert einer Option erfährt man durch `:set <option>?^M`.

Setzen

Booleans: `:set [no]<option>^M`

Strings: `:set <option>=<value>^M`

Einige Einstellungen

Option	Default	Beschreibung
autowrite	off	Speichern beim Dateiwechsel
number	off	Zeilennummerierung anzeigen
autoindent	off	automatisches Einrücken neuer Zeilen
shiftwidth	8	Länge einer Einrückung
tabstop	8	angezeigte Länge eines Tabs
expandtab	off	füge Leerzeichen statt Tabs ein (Vim)
filetype	–	Dateityp (Vim)

Speichern von Makros und Einstellungen

Bei jedem Start von Vi werden die Ex-Kommandos aus der Datei `~/.exrc` ausgeführt.

In diese Datei kann man einfach `map`-Befehle eintragen.

Vim-Erweiterungen

Vim führt außer der `~/.exrc` auch die `~/.vimrc` aus.

Hat man die Dateityp-Erkennung aktiviert, so wird zusätzlich `~/.vim/ftplugin/<type>.vim` geladen.

Hinweise

Scripting

Der Ursprüngliche Vi hat keine weiteren Scripting-Möglichkeiten.

Vim besitzt jedoch eine eigenen Skriptsprache und Bindings für die gängigsten Skriptsprachen.

Andere Vi-Varianten haben teilweise auch Unterstützung fürs Scripting.

Makro-Sammlungen

Im Internet finden sich Sammlungen von Makros, die beim editieren bestimmter Dateietypen nützlich sein können.

Vim-Plugins

Neben den mitgelieferten gibt es für Vim auch noch zusätzliche Filetype-Plugins, die meist mehr als nur ein paar Makros zur Verfügung stellen.

Tipps und Scripts: `http://www.vim.org/`

Vim-Latexsuite: `http://vim-latex.sf.net/`

Literatur

- Fred Buck: Vi Macros, Abbreviations and Buffers, `http://soma.npa.uiuc.edu/docs/vi.macros`
- Vim Online Hilfe, in Vim `:help`
- NVi Manpage: `nvi(1)`
- Wikipedia Eintrag zu Terminals, `http://en.wikipedia.org/wiki/Computer_terminal`