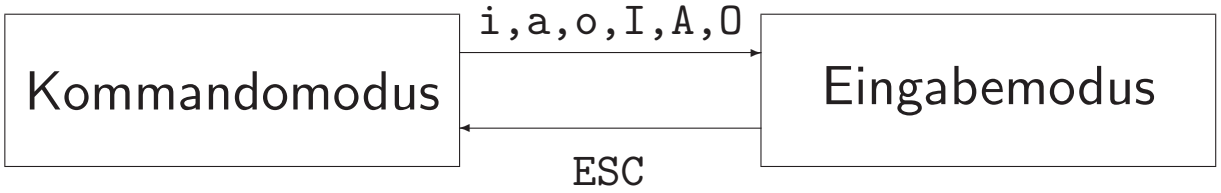
Der Standard-Editor vi

- unabhängig von Terminaltyp, Shell und Graphik
- Aufruf durch `vi` oder `vi` <*dateiname*>
- bearbeitet Kopie der Datei im Hauptspeicher
- zwei (drei) Betriebsmodi



Kommandomodus

- Manövrieren des Cursors mit Cursortasten oder

`h`	ein Zeichen nach links
`l`	ein Zeichen nach rechts
`j`	eine Zeile nach unten
`k`	eine Zeile nach oben
`0`	Zeilenanfang
`$`	Zeilenende
n`G`	in die n-te Zeile
`G`	in die letzte Zeile

- Löschen

`x`	Zeichen unter Cursor
`X`	Zeichen vor Cursor
`d0`	bis Zeilenanfang
`d$`	bis Zeilenende
`dd`	eine Zeile
`dG`	alle Zeilen ab aktueller Zeile
`d1G`	alle Zeilen bis zur aktuellen Zeile

Die Kommandos `x`, `X` und `dd` können durch Voranstellen einer Zahl (z.B. `3dd`) mehrfach angewendet werden (z.B. 3 Zeilen löschen).

- eine Auswahl weiterer Kommandos

`r`	Zeichen unter Cursor ersetzen
`R`	alle Zeichen ersetzen bis `ESC`
`J`	Zeilenumbruch beseitigen
`yy`	Zeile in Puffer speichern
`p`	Zeile im Puffer als nächste Zeile einfügen
`P`	Zeile im Puffer oberhalb einfügen
`:w` [*datei*]	Speichern [in *datei*]
`:r` *datei*	Einlesen von *datei*
`:q[!]`	Beenden [ohne Speichern]
`:!` *cmd*	Shell-Kommando *cmd* ausführen

Mehrfachausführung von `yy` möglich (*n*`yy`)

Die UNIX-Shells

- Aufgabe:
  - Kommunikation mit dem Benutzer (Schnittstelle Benutzer – System)
    - Eingabeprompt
    - Kommandointerpretation
    - Aufruf von Programmen oder Systemaufrufe
  - Verarbeitung von Kommandostapeln aus Shell-Skripten ("Programmiersprache")

- Arten:
  - `sh`: Bourne-Shell (Steve Bourne, *Bell Labs*)
  - `csh`: C-Shell (Bill Joy, *University of California*) $\rightsquigarrow$ `tcsh`: Tenex C-Shell
  - `bash`: Bourne-again-Shell (LINUX-Standard-Shell)
  - Korn-Shell `ksh`, Z-Shell `zsh`, Almquist-Shell `ash`, ...

In diesem Kurs: hauptsächlich `tcsh`.

Login- vs. Subshell

- Bei der Anmeldung eines Benutzers wird automatisch ein Shell-Prozess gestartet: die *Login-Shell*.

  $\rightsquigarrow$ in einer Terminal-Umgebung: Die Login-Shell wird im (einzigen) Terminalfenster ausgeführt. (keine Graphik!)

  $\rightsquigarrow$ bei graphischer Oberfläche: Die Login-Shell wird nicht angezeigt.
  In jedem Terminal, das als Fenster geöffnet wird, startet eine *Subshell*, die von der Login-Shell abstammt.

- Jede Subshell kann weitere Subshells erzeugen, z.B. durch
  - Aufruf eines Shellprogramms `/bin/sh`, `/bin/tcsh`, ...
  - Shellsprung,
  - Aufruf von anderen Programmen.

Anlaufdateien

- `.login` legt den Umgebungsbereich fest
- `.tcshrc` konfiguriert die Shell und modifiziert ggf. den Umgebungsbereich
- beim Start einer *Login-Shell*: `.login`, `.tcshrc`
- beim Start einer *Subshell*: `.tcshrc`
- zuvor können entsprechende Dateien aus `/etc` ausgeführt werden (Einstellungen für alle Benutzer)
- können die Ausführung weiterer Dateien bewirken (z.B. `.login.solaris`, `.cshrc`, `.aliases`)

Funktionsweise der Shell

1. Anlaufdateien abarbeiten
2. Prompt ausgeben, Eingabe lesen
3. Analyse der Kommandozeile
4. Kommando ausführen (als *shell builtin* oder ausführbare Datei)
5. *Exit-Status* speichern (wichtig für Shell-Programmierung),
   ggf. Fehlermeldungen in *stderr* schreiben;
   zu (2)

Shell- vs. Umgebungsvariablen

- **Shellvariablen**
  - lokale Variablen (nur in der aktuellen Shell gültig)
  - `set`
  - `set` *variable* `=` *wert*
  - `unset` *variable*
- **Umgebungsvariablen**
  - werden an Subshells vererbt
  - `env`
  - `setenv` *Variable Wert*
  - `unsetenv` *Variable*
- Anzeige eines einzelnen Variablenwertes: `echo $`*variable*

Einige (System-) Umgebungsvariablen

GROUP	Name der Benutzergruppe, zu der der Benutzer gehört
USER	Name des Benutzers
HOME	Login-Verzeichnis des Benutzers
TERM	Bezeichnung des Terminaltyps
PATH	Suchpfad für Programme und Dateien
SHLVL	Schachtelungstiefe der aktuellen Shell
MANPATH	Suchpfad für Manualpages
MAIL	Mailbox des Benutzers
PWD	aktuelles Verzeichnis
OSTYPE	Bezeichnung des Betriebssystems
HOST	Name des Rechners, auf dem die Shell läuft
TZ	lokale Zeitzone

Einige (System-) Shellvariablen

⋆	**group**	Name der Benutzergruppe, zu der der Benutzer gehört
⋆	**user**	Name des Benutzers
⋆	**home**	Login-Verzeichnis des Benutzers
⋆	**term**	Bezeichnung des Terminaltyps
⋆	**path**	Suchpfad für Programme und Dateien (*als Wortliste*)
⋆	**shlvl**	Schachtelungstiefe der aktuellen Shell
	cwd	aktuelles Verzeichnis
	owd	vorheriges Verzeichnis
	tty	Terminal, auf dem die Shell läuft
	prompt	Festlegung des Eingabeprompt
	history	Anzahl der gespeicherten Kommandos

⋆: direktes Analogon im Umgebungsbereich

Substitutionen

Auf der Kommandozeile werden eine Reihe von Substitutionen durchgeführt, bevor der eigentliche Programmaufruf erfolgt.

1. **Variablensubstitution**
   - \$*variable* wird durch den Wert von *variable* ersetzt
2. **Dateinamenexpandierung**
   - Wildcards (Joker) werden durch *passende* Dateinamen ersetzt
   - (versteckte) .-Dateien werden nicht in die Expandierung einbezogen
     $\rightsquigarrow$ gesonderte Expandierung von Mustern mit . als erstes Zeichen

Wildcard	Bedeutung
*	beliebeige Zeichenkette (auch die leere)
?	ein beliebeiges Zeichen
[...]	ein Zeichen aus der Menge, z.B. [aeiou]
[*von*-*bis*]	ein Zeichen zwischen *von* und *bis*
[^...]	ein Zeichen außerhalb der Menge, z.B. [^0-9]
^...	Negation der gesamten Wildcard
{*wort1*,...,*wortn*}	eines der Wörter
~	absoluter Pfadname des (eigenen) Login-Verzeichnisses
~*user*	abs. Pfadname des Login-Verzeichnisses von *user*

3. **Kommandosubstitution**

- Ein in ‘...‘ eingeschlossenes Kommando wird auf der Kommandozeile durch seine Ausgabe ersetzt.

4. **Historysubstitution** — einige Aspekte

- `history` zeigt die gespeicherte Befehlsliste an (shell-built-in)
- Bei der Historysubstitution wird auf Kommandos aus dieser Liste zurückgegriffen.
- Aufrufschema: `!`$event$`[:`$wort$`] [[:`$modifier$`] ...]`
  - $event$ bezeichnet Befehl (Event) aus der Liste
  - $wort$ bezeichnet Wörter des Events
  - $modifier$ legt die Art der Verwendung fest

    $\rightsquigarrow$ Standard: Event ausführen

z.B.:

event	Bedeutung
!!	letzter Befehl
!n	n-ter Befehl
!-n	n-ter Befehl vom Ende der Liste
!*string*	letzter Befehl, der mit *string* beginnt
!?*string*?	letzter Befehl, der *string* enthählt
wort	
n	n-tes Wort (0 = Kommandoname)
$	letztes Wort
*	alle Wörter außer Kommandoname
modifier	
p	gib Kommandozeile aus, ohne sie auszuführen

- Es gibt für Zugriffe auf Wörter des letzten Befehls Abkürzungen, z.B. `!*` für `!!:*`

5. **Aliassubstitution**

- `alias` zur Anzeige vereinbarter Alias-Kommandos
- `alias` $name\ kommando$
  $\rightsquigarrow$ bei Aufruf von $name$ wird $kommando$ ausgeführt
- Soll in $kommando$ eine Historysubstitution ausgeführt werden, so bezeichnet `!!` den Befehl, mit dem der Alias aufgerufen (verwendet) wurde.
  Das `!`-Symbol ist dabei zu quotieren.
- `unalias` $name$ zum Löschen des Aliaskommandos
- $\backslash name$ zum einmaligen Außerkraftsetzen des Alias

Reihenfolge der Substitutionen in der `tcsh`

1. Historysubstitution
2. Aliassubstitution
3. Variablensubstitution
4. Kommandosubstitution
5. Dateinamenexpandierung

Achtung: In einigen anderen Shells gibt es abweichende Reihenfolgen.

Apostrophiermechanismen/Quoting

- Zeichen mit Sonderbedeutung in der (t)csh (*Metazeichen*):

  `&` `|` `;` `<` `>` `(` `)` `[` `]` `{` `}` `‘` `’` `"` `!` `?` `*` `$` `~` `\` `&` `^` *space, tab, newline*

- Unterdrückung der Sonderbedeutung:

  `\` schützt das direkt folgende Zeichen,

  `'...'` schützt alle eingeschlossenen Zeichen,

  `"..."` schützt alle eingeschlossenen Zeichen außer `\`, `$`, `!` und ‘

  vor einer Interpretation durch die Shell.