1.7 Anwendung/Expansion und Substitution

1.7.1 Wie kann ich meine Arbeit vereinfachen?

Die Arbeit am System kann manchmal sehr zeitaufwändig sein. So gibt es Situationen wie:

- Alle doc-Dateien im Verzeichnis, deren Name mit `Backup` beginnt, sollen gelöscht werden
- Im persönlichen Verzeichnis von `meier` müssen Unterverzeichnisse `Service5000` bis `Service5999` angelegt werden
- An verschiedenen Stellen soll ein Unterverzeichnis mit dem Namen `InfoMaschine23876.238678236` angelegt werden

Wie können wir uns solche Arbeiten erleichtern? Die Shell bietet dazu zwei Mechanismen an,

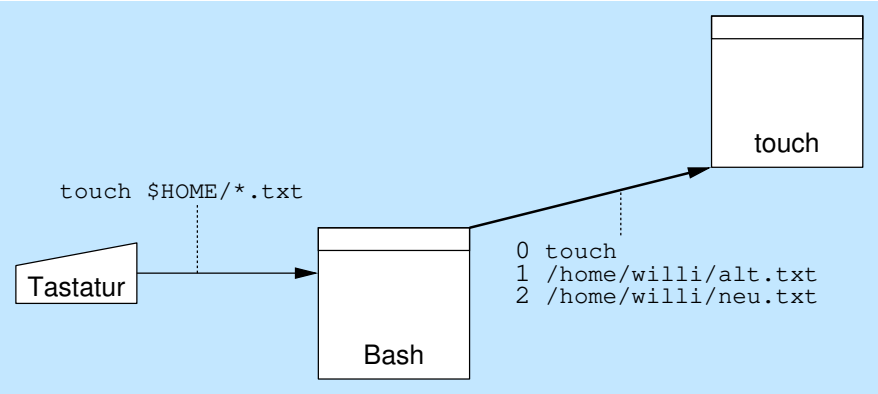

Abbildung 1: Veränderung der Befehlszeile

nämlich die *Substitution* und die *Expansion*. In beiden Fällen wird ein von uns auf der Befehlszeile eingegebener String nach festen Regeln verändert und die veränderte Befehlszeile ausgeführt (Abbildung 1).

Eine solche Veränderung wird allerdings immer nur dann vorgenommen, wenn in der Befehlszeile bestimmte Sonderzeichen wie etwa der Stern, das Dollarzeichen oder geschweifte Klammern vorhanden sind (Abbildung 2).

Der Unterschied zwischen Substitution und Expansion liegt einzig darin, dass die Substitution aus einem Wort ein anderes Wort macht, während die Expansion aus einem Wort (ein oder) mehrere andere Worte erzeugen kann.

1.7.2 Tilden-Substitution

Bei der Tilden-Substitution wird auf der Kommandozeile das Wort „~“ durch das Home-Verzeichnis des aktuellen Benutzers ersetzt. Man kann das Ergebnis sichtbar machen durch den Befehl `echo`:

Terminal

```
schueler@debian964:~$ echo  Mein Zuhause ist ~
Mein Zuhause ist /home/schueler
```

Möchte man bei einer allgemeinen Befehlszeile (ohne `echo`) die Substitution ansehen, setzt man den Bash-Schalter `set -x`. Dann verhält sich die Shell so, dass sie jede Befehlszeile erst auf den Bildschirm ausgibt, bevor sie sie ausführt:

Terminal

```
schueler@debian964:~$ set -x # Einschalten
schueler@debian964:~$ touch ~
+ touch /home/schueler
```

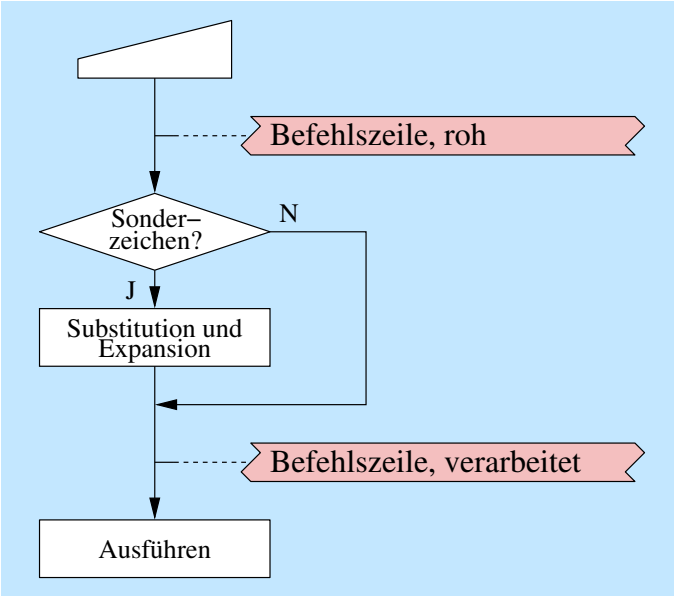

Abbildung 2: Ablauf der Veränderung

```
schueler@debian964:~$ set +x # Ausschalten
schueler@debian964:~$ touch ~
schueler@debian964:~$  # nix zu sehen
```

Das Home-Verzeichnis anderer Benutzer erhält man durch Anhängen des Benutzernamens an das Tildenzeichen:

Terminal
```
schueler@debian964:~$ echo Der Benutzer root wohnt in ~root
Der Benutzer root wohnt in /root
```

1.7.3 Klammern-Expansion von Listen

Mit der Klammern-Expansion (*brace expansion*) kann man aus wenigen Zeichen viel Ausgabe erzeugen. Aus einer durch Komma getrennten Liste von Wortteilen werden mehrere Wörter erzeugt.

Terminal
```
schueler@debian964:~$ echo Ausfahrt_Bielefeld-{Zentrum,Sennestadt}
Ausfahrt_Bielefeld-Zentrum Ausfahrt_Bielefeld-Sennestadt
```

Leerzeichen innerhalb von Worten müssen durch Anführungszeichen oder durch Backslash entschärft (maskiert) werden:

Terminal
```
schueler@debian964:~$ echo Ausfahrt_Herford-{Ost,"Bad Salzuflen"}
Ausfahrt_Herford-Ost Ausfahrt_Herford-Bad Salzuflen
schueler@debian964:~$ echo Ausfahrt_Herford-{Ost,Bad\ Salzuflen}
Ausfahrt_Herford-Ost Ausfahrt_Herford-Bad Salzuflen
```

Verwendet man mehrere Expansionen in einem gemeinsamen Ausdruck, werden alle möglichen Kombination erzeugt:

Terminal
```
schueler@debian964:~$ echo Abfahrt_{Herford,Minden,Detmold}-{Ost,West}
Abfahrt_Herford-Ost Abfahrt_Herford-West Abfahrt_Minden-Ost
Abfahrt_Minden-West Abfahrt_Detmold-Ost Abfahrt_Detmold-West
```

```
schueler@debian964:~$ echo {1,2,3{a,b}}.doc
1.doc 2.doc 3a.doc 3b.doc
```

Die Klammern-Expansion von Listen unterstützt auch Bereiche:

Terminal
```
schueler@debian964:~$ echo {A..C}
A B C
schueler@debian964:~$ echo {8..12}
8 9 10 11 12
schueler@debian964:~$ echo {08..12}
08 09 10 11 12
```

1.7.4 Wildcard-Expansion von Pfadnamen

1.7.4.1 Allgemein Auch bei der Wildcard-Expansion von Pfadnamen (englisch *filename globbing*) kann aus einer kurzen Zeichenkette viel Text entstehen; hier wird jedoch aus einem Verzeichnisinhalt (Liste aller Einträge) die Menge aller Namen herausgefiltert, die zu einem Muster passt:

Terminal
```
schueler@debian964:~$ touch 1.doc willi.doc hallo.doc sonst.txt dt_txt
schueler@debian964:~$ echo *
1.doc dt_txt hallo.doc sonst.txt willi.doc
schueler@debian964:~$ echo *.*
1.doc hallo.doc sonst.txt willi.doc
schueler@debian964:~$ echo ??ll?.doc
hallo.doc willi.doc
schueler@debian964:~$ echo *o*
1.doc hallo.doc sonst.txt willi.doc
```

Hierbei steht also der Stern für eine beliebig lange (auch leere) Zeichenkette, das Fragezeichen für genau ein Zeichen[1][2]

Mit dem Befehl `set -x` kann man sichtbar machen, was passiert:

Terminal
```
schueler@debian964:~$ set -x
schueler@debian964:~$ echo *o*
+ echo 1.doc hallo.doc sonst.txt willi.doc
1.doc hallo.doc sonst.txt willi.doc
```

Es ist also wieder die Shell, die die Expansion vornimmt und den `echo`-Befehl mit der expandierten Argumentliste startet. Mit dem Befehl `set +x` macht man den obigen `set`-Befehl wieder rückgängig.

Wenn zu einem Suchmuster kein Eintrag gefunden wird, wird nicht expandiert, stattdessen wird das Suchmuster in den Befehl eingesetzt:

Terminal
```
schueler@debian964:~$ echo *.txt
sonst.txt
schueler@debian964:~$ echo *.tex
*.tex
```

[1] Anders als bei regulären Ausdrücken muss das Muster nicht auf irgendeine Stelle im Text passen, sondern auf den gesamten Eintragsnamen. `*o*` findet also alle Einträge mit einem `o` irgendwo im Namen.

[2] Der Punkt im Eintragsnamen hat hier keine besondere Bedeutung, im Gegensatz zu FAT mit seinen 8+3 Zeichen langen Datei- und Verzeichnisnamen. Deshalb wird mit `*.*` der Dateiname `datei_txt` nicht gefunden.

1.7.4.2 Wildcard-Zeichen Außer * und ? gibt es in Linux noch weitere Wildcard-Zeichen: Zwei oder mehr Zeichen in eckigen Klammern meinen eine entweder-oder-Auswahl. `[Hh]einz` passt auf `Heinz` und `heinz`. `Test[ABC]` passt auf `TestA`, `TestB` und `TestC`.

Mit Hilfe des Minuszeichens (=Bindestrichs) kann man in den eckigen Klammern jeweils einen Bereich eingeben: `IFS[1-3]A` passt auf `IFS1A`, `IFS2A` und `IFS3A`. Genauso passt `[a-z]` auf alle Buchstaben zwischen `a` und `z`. Welche das sind, hängt aber von der eingestellten Sortierreihenfolge ab, und die ist von Land zu Land unterschiedlich. ASCII sortiert ABC…Z…abc…Z, in Deutschland sortiert man aAäÄbBcC…zZ:

Terminal
```
schueler@debian964:~$ ls *
apfel.ods dt_txt  hallo.doc IFS3A.xls
schueler@debian964:~$ ls [A-Z]*
dt_txt  hallo.doc # wo ist IFS3A.xls?
```

Man kann diese Sortierreihenfolge auch abschalten. Dazu gibt es die Bash-Option `globasciiranges`:

Terminal
```
schueler@debian964:~$ shopt -s globasciiranges # Oder: LC_COLLATE=C
schueler@debian964:~$ ls [A-Z]*
IFS3A.xls # so wie erwartet
schueler@debian964:~$ shopt +s globasciiranges # Umschaltung zurück
```

Deshalb verwendet man anstelle von Bereichen besser sogenannte Zeichenklassen: `[[:lower:]]` meint alle Kleinbuchstaben des aktuellen Zeichensatzes:

Terminal
```
schueler@debian964:~$ touch anton Berta caesar doris Erna 33604
schueler@debian964:~$ echo [[:lower:]]*
anton caesar doris
```

Den Bereich, den eine Zeichenklasse angibt, kann man auch für eine Suche ergänzen:

Terminal
```
schueler@debian964:~$ echo [[:lower:]123]*
33604 anton caesar doris
```

Zeichenklasse	Inhalt
upper	Ein Großbuchstabe
alpha	Ein Buchstabe
digit	Eine Ziffer
xdigit	Eine Hexadezimalziffer
alnum	Eine Ziffer oder ein Buchstabe
punct	Ein Satzzeichen, aber kein Leerzeichen
space	Ein Leerzeichen, Tabulator, Zeilenumbruch o. ä.

Tabelle 1: Zeichenklassen (Auswahl)

Weitere wichtige Zeichenklassen zeigt Tabelle 1.

1.7.4.3 Option oder Dateiname? Ein Problem, das nicht nur mit Wildcard-Expansion zu tun hat, ist die Tatsache, dass viele Befehle Optionen annehmen, die mit dem Minuszeichen eingeleitet werden, z.B. `ls -l`. Wenn jetzt eine Datei `-l` heißt, wird es schwierig, sie zu bearbeiten, weil der Befehl (z.B. `rm`) das Kommandozeilenargument `-l` für eine Option hält:

Terminal
```
schueler@debian964:~$ rm -l
rm: invalid option -- 'l'
Versuchen Sie "rm ./-l", um die Datei "-l" zu entfernen.
```

Folgende zwei Lösungen sind möglich: Die letzte Option eines Befehls vor der Liste mit Dateinamen heißt stets `--`. Alles, was danach folgt, ist ein Dateiname:

Terminal
```
schueler@debian964:~$ rm -- -l
```

Eine weitere Möglichkeit ist es – wie in der Fehlermeldung vorgeschlagen – vor dem Dateinamen den relativen Pfadnamen `"."` anzugeben:

Terminal
```
schueler@debian964:~$ rm ./-l
```

Eine Löschen aller Dateien (einschließlich der Datei mit dem Namen `-l`) im aktuellen Verzeichnis sieht also so aus:

Terminal
```
schueler@debian964:~$ rm -- *
+ rm -- -l 1.doc dt_txt hallo.doc sonst.txt willi.doc
```

1.7.4.4 Versteckte Dateien Unter Linux gelten Dateien und Verzeichnisse, deren Namen mit Punkt beginnen, als versteckt. Sie werden durch normale Befehle nicht angezeigt (damit ihr Vorhandensein nicht nervt) und bearbeitet (damit sie nicht aus Versehen gelöscht oder verschoben werden). Besonders Konfigurationsdateien sind meistens versteckt.

Daher werden sie auch durch Wildcard-Expansion nicht mit expandiert. Das kann man jedoch ändern:

- Mit dem Muster `.*` werden alle versteckten Dateien und Verzeichnisse gefunden; mit dem Muster `.[^.]*` werden alle versteckten Dateien und Verzeichnisse außer `.` und `..` gefunden (sehr sinnvoll!).
- Mit dem Befehl `shopt -s dotglob` kann man die Ausnahmebehandlung für die versteckten Einträge abschalten (und mit `shopt -u dotglob` wieder einschalten).

1.7.4.5 Keine Expansion, bitte! Wie löscht man eine Datei mit dem Namen `*.*`? Wenn man nicht möchte, dass ein Wildcard-Zeichen interpretiert wird, muss man es maskieren. Drei Arten sind möglich:

a) vor jedes Sonderzeichen ein Backslash `\` stellen: `rm \*.\*`

b) den gesamten Text in einfache Hochkommata `'` einfassen: `rm '*.*'`

c) den gesamten Text in Anführungszeichen `"` einfassen: `rm "*.*"`

Als weitere Lösung bietet sich an, mit dem Befehl `set -f` die Wildcard-Expansion von Pfadnamen komplett abzuschalten:

Terminal
```
schueler@debian964:~$ set -f
schueler@debian964:~$ rm *.*
schueler@debian964:~$ set +f
```

Mit `set +f` schaltet man anschließend wieder zurück.

1.7.5 Variablen-Substitution

1.7.5.1 Eigene Shell-Variablen Mit der Variablen-Substitution (auch *parameter expansion* genannt) erhält die Shell eigene Variablen, wie man sie aus Programmiersprachen kennt. Sie heißen *Shell-Variablen.*

Terminal
```
schueler@debian964:~$ ESSEN="Quark mit Soße"
schueler@debian964:~$ echo "$ESSEN"
Quark mit Soße
```

Beim Setzen des Inhalts (erste Zeile) steht der Name links vom Gleichheitszeichen, der Inhalt rechts in Anführungszeichen. Links und rechts vom Gleichheitszeichen dürfen keine Leerzeichen und keine Tabulatoren eingefügt werden.

Bei der Ausgabe (zweite Zeile) muss ein Dollar-Zeichen vor den Namen geschrieben werden. Dann wird der Name der Variablen durch den Inhalt der Variablen ersetzt.

1.7.5.2 Eingebaute Shell-Variablen Einige Shell-Variablen sind bereits gesetzt, wenn man eine Shell aufruft. Beispiele:

- `$PS1` ist das Aussehen der Eingabeaufforderung[3].
- `$PATH` ist die Liste der Verzeichnispfade, in denen nach Programmen gesucht wird, wenn man einen Befehlsnamen eingibt.

1.7.5.3 War das schon alles? In Shellskripten sind Variablen sehr wichtig. Deshalb findet man beim Thema *Variablen in Shellskripten* noch weitere Informationen zur Variablen-Substitution.

1.7.6 Weitere Substitutionen

- Kommando-Substitution — folgt noch
- Arithmetische Substitution — folgt noch (beim Thema *Variablen in Shellskripten*
- History-Substitution — selten benötigt
- Prozess-Substitution — siehe:
  - Holger Trapp: Die GNU-Shell Bash
    `http://www-user.tu-chemnitz.de/~hot/unix_linux_werkzeugkasten/bash.html`
  - *advanced bash scripting guide*
    `http://tldp.org/LDP/abs/html`

[3] Bei Wind. heißt diese Variable `PROMPT`.