Propädeutikum
Programmierung in der Bioinformatik

Fortsetzung Shell

Thomas Mauermeier

23.10.2018

Ludwig-Maximilians-Universität München

Wildcards?

Kartenspiel

Karte die als beliebige Karte gespielt werden kann

Informatik

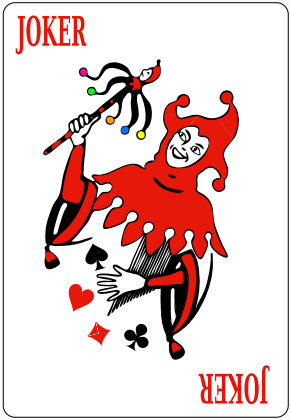
Platzhalter für beliebige (Menge von) Zeichen

Beispiel: Shell

Wildcard "`*`": Matcht irgendeine Zeichenfolge

`rm *.jpg` löscht alle Files mit der Endung `.jpg`

Achtung: Wildcards in Shell $\neq$ Reguläre Ausdrücke!



Wildcards in Bash

Wildcard	Match
`*`	beliebige Zeichenfolge (0–∞ viele Zeichen)
`?`	beliebiges Zeichen (genau 1)
`[<chars>]`	beliebiges Zeichen aus `<chars>` (genau 1)
`[!<chars>]`	beliebiges Zeichen das *nicht* in `<chars>` ist (genau 1)
`[[:<class>:]]`	beliebiges Zeichen aus vordefinierter Klasse (genau 1)

Ranges – Angabe als Bereich

`[<von>-<bis>]`

z.B. `[a-z]`, `[0-9]`, `[A-Z]`

abgesehen von speziellen Ranges wie `[d-o]` praktisch identisch zu Klassen

Klasse	Match
`alpha`	alphabetische Zeichen
`digit`	numerische Zeichen
`alnum`	alphanumerische Zeichen
`lower`	lowercase Zeichen (“Kleinbuchstaben”)
`upper`	uppercase Zeichen (“Großbuchstaben”)

Beispiele

Wildcard	Match
`*`	Alle Dateien
`*.jpg`	Dateien mit der Endung `.jpg` (⚠ Case sensitive!)
`*.[jJ][pP][gG]`	Dateien mit Endung `.jpg` (Case insensitive)
`foo.???`	Dateien die mit `foo.` beginnen & exakt 3 Zeichen langer Endung
`foo[[:digit:]]`	`foo0`, `foo1` ... `foo9` (aber z.B. *nicht* `foo10`)
`foo[0-9][0-9]`	`foo00` bis `foo99` (aber z.B. *nicht* `foo1`)
`foo[A-Z]`	`fooA` bis `fooZ` (aber z.B. *nicht* `fooa`)
`foo[[:upper:]]`	`fooA` bis `fooZ` (aber z.B. *nicht* `fooa`)
`[!abc]*`	Dateien die *nicht* mit a, b, c beginnen, dafür aber z.B. `dattel`
`[abc]*`	Nur Dateien die mit a, b, c beginnen, aber z.B. *nicht* `dattel`
`[![:lower:]]*`	Alle Dateien die *nicht* mit einem Kleinbuchstaben beginnen

Wildcard Expansion – abhängig von vorhandenen Dateien

```
mustermann@hallimasch:~/demo $ echo foo[1-9]
foo1 foo2 foo3
```

Warum ist Ausgabe nicht: `foo1 foo2 ... foo9`?

```
mustermann@hallimasch:~/demo $ ll
total 0
-rw-r--r-- 1 mustermann stud 0 Okt 25 22:12 foo1
-rw-r--r-- 1 mustermann stud 0 Okt 25 22:12 foo2
-rw-r--r-- 1 mustermann stud 0 Okt 25 22:12 foo3
```

- Wildcard "gleicht ab" welche Files im angegebenen Pfad in Pattern passen
  - Liste der Filenames die zu Pattern passen wird aufgebaut
  - `foo[1-9]` "expandiert" zu String: `foo1 foo2 foo3`

Brace Expansion

Syntax

{<von>..<bis>}

- <von> und <bis> können Buchstabe oder Zahl sein
- Expandieren zu String *unabhängig* von vorhandenen Dateien

```
mustermann@hallimasch:~/demo $ echo {a..c}{00..03}
a00 a01 a02 a03 b00 b01 b02 b03 c00 c01 c02 c03
```

```
mustermann@hallimasch:~/demo $ mkdir vorlesung{1..3} && ls -l
total 20
drwxrwxr-x 2 mustermann stud 4096 Okt 25 22:12 vorlesung1
drwxrwxr-x 2 mustermann stud 4096 Okt 25 22:12 vorlesung2
drwxrwxr-x 2 mustermann stud 4096 Okt 25 22:12 vorlesung3
```

Standard I/O Streams & Umleitungen

```
mustermann@hallimasch:~/demo $ echo "Warum ist alles so kompliziert!"
Warum ist alles so kompliziert!
```

I/O Streams:

stdin standard input – Kanal für Eingaben
stdout standard output – Kanal für Ausgaben
stderr standard error – Kanal für Fehlermeldungen

Umleitungen:

`<cmd> > <file>`	stdout von `<cmd>` überschreibt (⚠) `<file>`
`<cmd> >> <file>`	Konkateniert stdout von `<cmd>` an Ende von `<file>`
`<cmd> 2> <file>`	stderr von `<cmd>` wird in `<file>` geschrieben
`<cmd> < <file>`	stdin von `<cmd>` wird aus `<file>` gelesen

Pipes – weitergeben von Output

$$f \circ g \quad \approx \quad \texttt{g | f}$$

$$f : \text{stdin} \to \text{stdout}$$

$$g : \text{stdin} \to \text{stdout}$$

$$f \circ g : \text{stdin} \xrightarrow{g} \text{stdout} \xrightarrow{f} \text{stdout}$$

Beispiel: Zählen von Files

```
ls -l | tail -n+2 | wc -l
```

- `ls -l` gibt Ordnerinhalt in Listenform aus
- `tail -n+2` schneidet allen Output vor Zeile 2 ab so dass nur Files in Output
- `wc -l` zählt die übrigen Lines

Umleitungen vs. Pipes

Umleitung: `befehl1 > file && befehl2 < file`

Langsamer, Zwischenschritt aber für spätere Verwendung gespeichert

Pipe: `befehl1 | befehl2`

Schneller, Zwischenschritt aber nur temporär gespeichert

grep – Suche nach Mustern

grep <muster> <file>

<muster>

Regulärer Ausdruck (mehr in späterer VL)

Für Heute: <muster> = Suchbegriff

`^`	markiert Zeilenanfang
`$`	markiert Zeilenende
`-v`	Alles was *nicht* auf Muster passt
`-w`	Nur ganze (whole) Wörter matchen

<file>

Pfad:

```
grep "foo" path/to/file
```

oder stdin:

```
cat path/to/file | grep "foo"
```

`tar` und `gzip` – Packen, Entpacken und Komprimieren

`tar <flag> <pfad>`

für `.tar`-Files

Flag	Bedeutung
`-c`	**c**reate archive
`-x`	e**x**tract archive
`-t`	lis**t** contents
`-f`	**f**ile (Ziel)

Beispiel

```
tar -cf archiv zu_archivierendes
tar -xf archiv
tar -tf archiv
```

`gzip <flag> <pfad>`

für `.gz`-Files

Flag	Bedeutung
(leer)	Komprimiere file
`-d`	**d**ecompress file

Beispiel

```
gzip zu_komprimierendes
gzip -d zu_dekomprimierendes
```