1.9 Anwendung/Kommandosubstitution

1.9.1 Probleme beim Zusammenspiel zweier Programme

Folgendes Problem liegt vor: Alle Dateien und Verzeichnisse, die älter als zwei Tage sind, sollen mit dem Befehl `touch` auf das aktuelle Datum gebracht werden.

In diesem Beispiel-Fall sind aus Platzgründen im aktuellen Verzeichnis und darunter nur zwei Dateien vorhanden, die `x1` und `x2` heißen.

Der erste Versuch dazu läuft so:

Terminal

```
schueler@debian964:~$ ls -l
-rw-rw---- 1 schueler schueler 19 23. Apr 15:21 x1
-rw-rw---- 1 schueler schueler 31 25. Apr 15:30 x2
schueler@debian964:~$ find . -mtime +2 | touch
touch: Fehlender Dateioperand
"touch --help" liefert weitere Informationen.
```

Was ist passiert? Abbildung 1 gibt einen Hinweis.

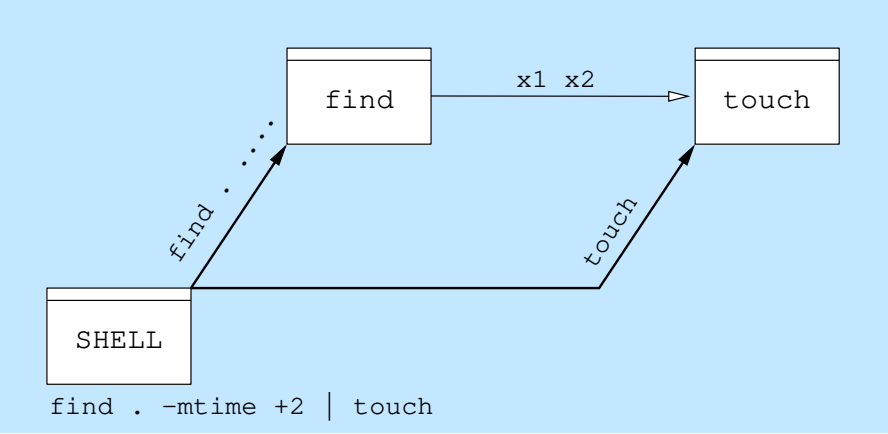

Abbildung 1: Erster Versuch, erfolglos

- `find` findet diese Dateien und gibt sie auf die Standardausgabe (=Bildschirmausgabe, dünner Pfeil). `touch` bekommt sie auf der Standardeingabe (=Tastatureingabe).
- Die Manual-Page zu `touch` sagt aber:

  ```
  BEZEICHNUNG
         touch - Zeitstempel von Dateien ändern
  ÜBERSICHT
         touch [OPTION]... DATEI...
  ```

  `touch` erwartet die Dateien nicht auf der Standardeingabe (=von der Tastatur), sondern als Parameter auf seiner Kommandozeile (dicker Pfeil unten rechts).

Kurz gefasst: Mit der Befehlsverkettung bekommt das Programm `touch` seine Daten auf der falschen Schnittstelle. Es handelt sich also um ein Schnittstellen-Problem, wie man es auch aus der Elektronik kennt: Das Signal kommt nicht so oder dort heraus, wo oder wie man es braucht. Deshalb verwendet man in diesem Fall einen Adapter.

1.9.2 Kommandosubstitution als Lösung

Der Adapter in unserem Fall heißt Kommando-Substitution. Diese Substitution wird mit dem Dollarzeichen und **einem** Paar runder Klammern eingeleitet:

Terminal
```
schueler@debian964:~$ touch $(find -mtime +2)
schueler@debian964:~$ ls -l
-rw-rw---- 1 schueler schueler 19 16. Mai 21:49 x1
-rw-rw---- 1 schueler schueler 31 16. Mai 21:49 x2
```

Es hat also funktioniert. Aber was ist bei dieser Kommando-Substitution passiert? Abbildung 2 zeigt den Ablauf.

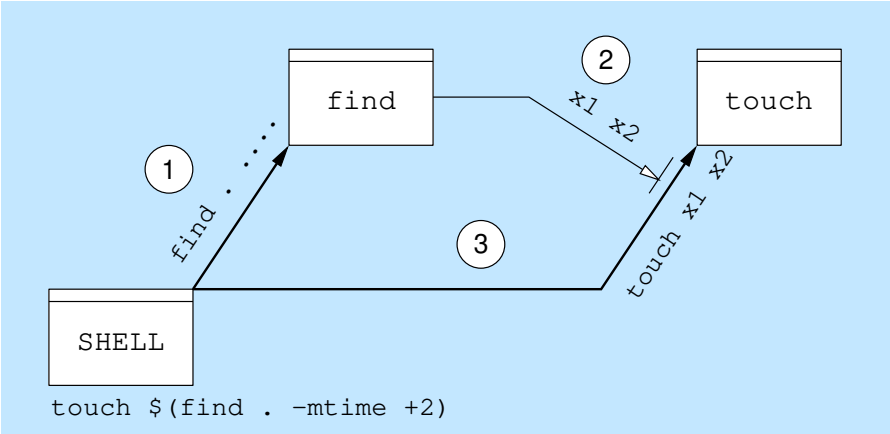

Abbildung 2: Schritte bei der Kommandosubstitution

- Schritt 1: Der Befehl in Klammern (hier `find . -mtime +2`) wird ausgeführt.
- Schritt 2: Das Ergebnis (hier `x1 x2`) wird an *die* Stelle der Kommandozeile kopiert, an der der Substitutionsbefehl, also der `$()`-Ausdruck gefunden wurde.
- Schritt 3: Die neue Kommandozeile mit dem ersetzten Text wird ausgeführt.

Ein möglicher Merksatz: Bei der Kommandosubstitution wird eine Befehlszeile durch ihre Bildschirmausgabe ersetzt.

1.9.3 Hilfen

Mit der Bash-Option `set -x` kann man den Ablauf sichtbar machen. Mit dieser Option wird nämlich jede Befehlszeile einmal angezeigt, bevor sie ausgeführt wird:

Terminal
```
schueler@debian964:~$ set -x # Shell-Option setzen
schueler@debian964:~$ touch $(find . -name 'x?')
++ find . -name 'x?'
+ touch x1 x2
schueler@debian964:~$ set +x # Shell-Option loeschen
```

Die Zeile mit den zwei Pluszeichen wird zuerst ausgeführt (die zwei Pluszeichen zeigen, dass sie für eine andere Zeile ausgewertet wird). Die Zeile mit einem Pluszeichen enthält schon die beiden eingesetzten Dateinamen. Sie wird danach ausgeführt. Mit `set +x` wird die Shell-Option wieder gelöscht.

Abbildung 3 zeigt noch einmal den Datenfluss bei einer Kommandosubstitution.

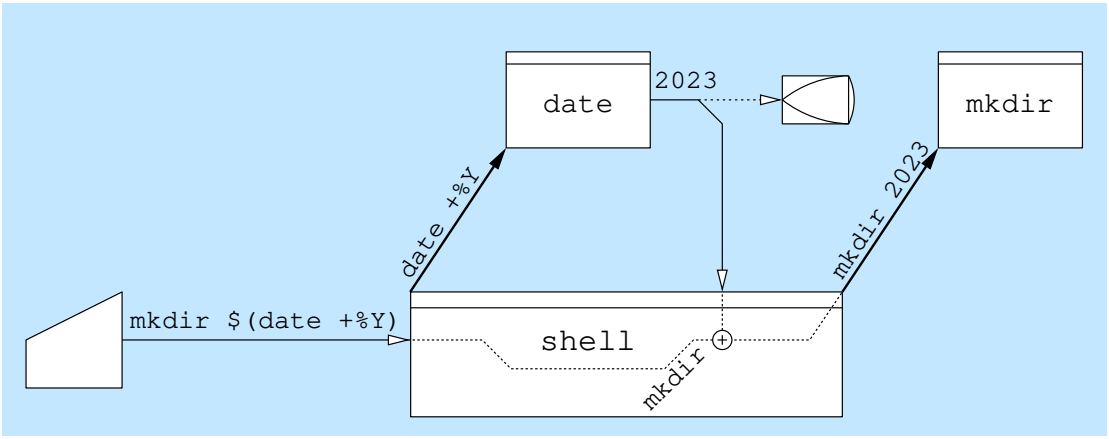

Abbildung 3: Datenfluss bei der Kommandosubstitution

1.9.4 Mehrere Kommandosubstitutionen in einer Zeile

Man kann in derselben Befehlszeile mehrere Kommandosubstitutionen parallel unterbringen:

Terminal
```
schueler@debian964:~$ set -x
schueler@debian964:~$ mkdir "$(logname)_$(date +%Y)"
++ logname
++ date +%Y
+ mkdir schueler_2023
schueler@debian964:~$ set +x
```

Hier wurden zuerst `logname` und `date` ausgeführt, anschließend die Ergebnisse eingesetzt und zum Schluss `mkdir` ausgeführt. Die Anführungszeichen sind hier sinnvoll, weil der `date`-Befehl je nach Parametern auch Leer- und Sonderzeichen enthalten kann. Da das Anführungszeichen die $-Zeichen nicht maskiert, bleiben die Kommandosubstitutionen erhalten.

Außerdem ist es möglich, in derselben mehrere Kommandosubstitutionen ineinander zu schachteln:

Terminal
```
schueler@debian964:~$ set -x
schueler@debian964:~$ stat $(dirname $(which cal))
+++ which cal
++ dirname /usr/bin/cal
+ stat /usr/bin
 Datei: /usr/bin
 Größe: 131072 .....
schueler@debian964:~$ set +x
```

a) Hier wird zuerst `which cal` aufgerufen und gibt `/usr/bin/cal` aus.

b) Anschließend wird `dirname /usr/bin/cal` aufgerufen und gibt `/usr/bin` aus.

c) Zum Schluss erfolgt der Aufruf von `stat /usr/bin`.

1.9.5 Ältere Schreibweise

Für die Kommandosubstitution gibt auch eine ältere Schreibweise:

Terminal
```
schueler@debian964:~$ touch `find -type f -mtime +2`
```

Die umgekehrten Anführungszeichen, die man hier braucht, nennt man *backticks*. Man erhält sie auf der PC-Tastatur mit [Shift ⇑] - [Apostroph] . Diese Schreibweise ist allerdings nicht geeignet, verschachtelte Kommandosubstitutionen zu beschreiben, denn die Backticks dienen sowohl als öffnende als auch als schließende Klammer.

1.9.6 Wenn die Kommandosubstitution fehlschlägt

a) Zu lange Ersetzung
Es kann sein, dass der folgende Befehl auf einem größeren System nicht funktioniert.

Terminal
```
schueler@debian964:~$ stat $(find /)
... nach einigen Sekunden ...
Befehlszeile zu lang
```

Die Befehlsausgabe, die auf die Kommandozeile kopiert werden soll (hier die Ausgabe von `find`), kann (je nach Aufgabenstellung) eventuell sehr groß sein (z.B. mehrere MiByte). Die Länge von Befehlszeilen ist aber naturgemäß in jedem Betriebssystem begrenzt (z.B. 2 MiByte, herauszufinden mit dem Befehl `xargs --show-limits`). Die Shell weigert sich dann, die Substitution auszuführen.

b) Eine ersetzung
Eine weitere Fehlermöglichkeit ist hier:

Terminal
```
schueler@debian964:~$ grep Obersee $(find / -name "*.gibsnicht")
... warten und warten ...
 [Strg] + [C]
^C
schueler@debian964:~$ set -x
schueler@debian964:~$ grep Obersee $(find / -name "*.gibsnicht")
++ find . -name '*.gibsnicht'
+ grep --color=auto Obersee
 [Strg] + [C]
^C
schueler@debian964:~$ set +x
```

Hier liegt das Problem vor, dass das Kommando ins nichts ersetzt wird, weil `find` nichts findet und deshalb keine Bildschirmausgabe hat. Dadurch hat `grep` nur einen einzigen Parameter; wenn `grep` mit nur einem Parameter aufgerufen wird, wartet es auf Eingabe von der Tastatur, es arbeitet dann als Filterprogramm.

c) Ersetzung mit Leer- und Sonderzeichen
Auch dies kann passieren:

Terminal
```
schueler@debian964:~$ ls
-rw-rw---- 1 schueler schueler 270 3. Apr 17:00 staedte in owl.txt
-rw-rw---- 1 schueler schueler 613 3. Apr 17:00 staedte in nds.txt
schueler@debian964:~$ grep Bielefeld $(find . -name "*.txt")
rep: ./staedte: Datei oder Verzeichnis nicht gefunden
grep: in: Datei oder Verzeichnis nicht gefunden
grep: nds.txt: Datei oder Verzeichnis nicht gefunden
grep: ./staedte: Datei oder Verzeichnis nicht gefunden
grep: in: Datei oder Verzeichnis nicht gefunden
grep: owl.txt: Datei oder Verzeichnis nicht gefunden
schueler@debian964:~$ grep Bielefeld "$(find . -name "*.txt")"
```

```
grep: ./staedte in nds.txt
./staedte in owl.txt: Datei oder Verzeichnis nicht gefunden
```

Hier liegt der Fehler darin, dass die gefundenen Dateinamen Leer- oder Sonderzeichen enthalten. Wenn man die gesamte Kommandosubstitution mit Anführungszeichen maskiert, dann bekommt man nur ein Wort anstelle der vielen Dateinamen.

1.9.7 `xargs` statt Kommando-Substitution

Das Programm `xargs` ist eine Alternative zur Kommando-Substitution.

Terminal

```
schueler@debian964:~$ find . -type f | xargs stat
```

Was passiert hier?

a) Der Befehl vor dem Pipe-Symbol (`find /`) wird ausgeführt und gibt seine Ergebnisse (Dateiliste) aus.

b) `xargs` nimmt Worte von der Standardeingabe an.

c) `xargs` führt *den* Befehl (`stat`) aus, der in seinen eigenen Parametern angegeben ist, und fügt die eingelesenen Worte als weitere Parameter hinzu.

d) `xargs` wiederholt das so oft, bis alle Worte von der Standardeingabe abgearbeitet wurden. Danach beendet es sich.

Abbildung 4 zeigt den Datenfluss bei Verwendung von `xargs`. Im Unterschied zur direkten

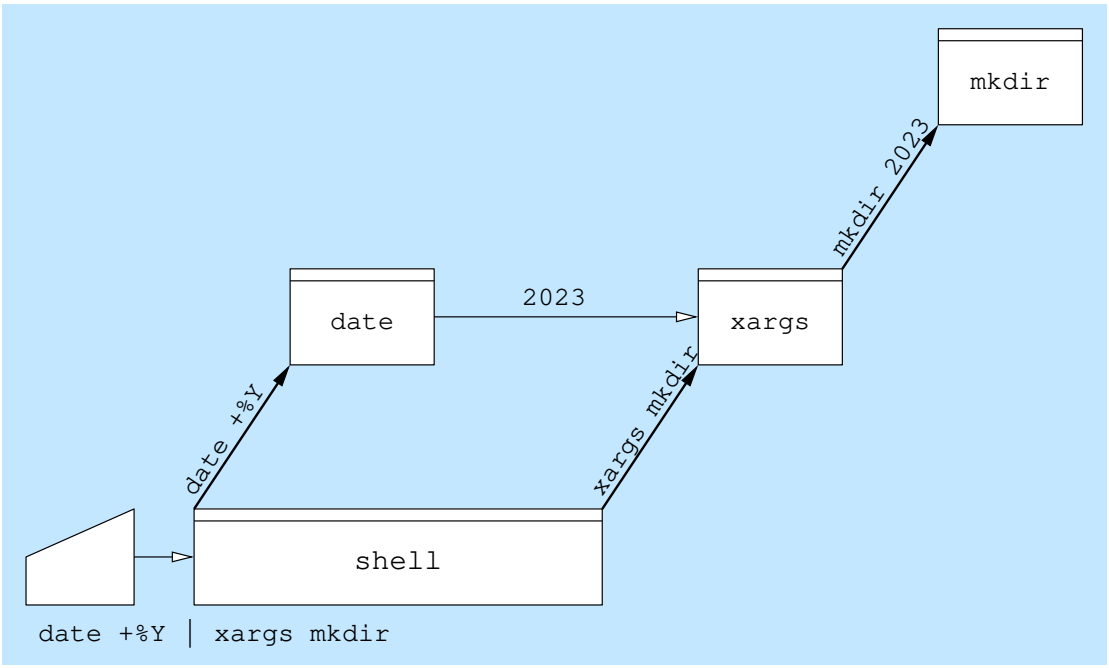

Abbildung 4: Datenfluss bei der Verwendung von `xargs`

Kommando-Substitution erhält `touch` von `xargs` nur so viele Parameter, wie das System verkraftet. Eine zu große Befehlszeile wie bei der Kommandosubstitution kann bei `xargs` daher nicht auftreten.

1.9.8 Optionen für `xargs`

Das Programm `xargs` erlaubt viele Optionen. Damit ist es – im Gegensatz zur Kommandosubstitution – konfigurierbar. Hier sind die wichtigsten Optionen:

- `xargs -n1 befehl`
  Für jedes Wort von der Standardeingabe wird der Befehl neu mit diesem Wort als einzigem Parameter aufgerufen. Manche Befehle (wie `dirname` oder `basename`) verkraften nur einen einzigen Parameter, für diese ist das gedacht. Hier ein Beispiel:

  Terminal
  ```
  schueler@debian964:~$ find /tmp -type f -name "*.txt" \
     | xargs -n1 basename
  fluesse.txt
  grepbeispiele.txt
  staedte.txt
  ```

- `xargs -r befehl` (oder `--no-run-if-empty befehl`)
  Falls die Liste leer ist, wird der Befehl nicht mit einer leeren Liste aufgerufen, sondern gar nicht. Dies ist sinnvoll für Befehle wie `grep`, die sich bei einer leeren Dateiliste ungewohnt verhalten: Sie erwarten dann einen Eingabedatenstrom von der Tastatur (Standardeingabe) und warten und warten und warten .... Hier wieder ein Beispiel:

  Terminal
  ```
  schueler@debian964:~$ find /tmp -name "*.gn" | xargs -r grep "see"
  schueler@debian964:~$  # Befehl nach Ende von find sofort beendet
  ```

- `xargs -0 befehl`
  Hier erkennt `xargs` den Beginn einen neuen Eingabewortes nicht am Leerzeichen, sondern am Stringendezeichen ASCII-0 (in C: `'\0'`, auch Terminator genannt). Dies ist unbedingt zu empfehlen für alle Fälle, in denen Datei- oder Verzeichnisnamen Leerzeichen, Zeilenendezeichen, Tabulatorzeichen und weitere Satzzeichen enthalten dürfen.

  Durch die Option `-0` werden solche Dateinamen nicht auseinandergerissen (das Stringendezeichen ASCII-0 ist das einzige Zeichen, das in einem Pfadnamen nicht vorkommen kann). Wenn man diese Option von `xargs` zusammen mit `find` benutzt, muss man bei find die entsprechende Option ebenfalls benutzen. Sie lautet dort `find -print0` und meint „Trenne die Liste der Namen durch ASCII-0“. Und hier das Beispiel:

  Terminal
  ```
  schueler@debian964:~$ ls
  -rw-rw---- 1 schueler schueler 270 3. Apr 17:00 staedte in owl.txt
  -rw-rw---- 1 schueler schueler 613 3. Apr 17:00 staedte in nds.txt
  schueler@debian964:~$ find . -name "*.txt" -print0 | \
     xargs -0 grep Bielefeld
  ./staedte in owl.txt:Bielefeld
  ```

- `xargs -0 -I ERSATZ befehlszeile`
  Hier setzt `xargs` das Eingabewort (ohne die Option `-0`: die Eingabewortliste) ein an jeder Stelle der Befehlszeile, in der der Begriff `ERSATZ` vorkommt:
  `find . -print0 | xargs -0 -I ERSATZ mv ERSATZ ERSATZ.bak`

  In diesem Beispiel wird jedes gefundene Dateisystemobjekt so umbenannt, dass an seinen Namen die Endung `.bak` angehängt wird.

  Die alte, aber immer noch oft benutzte Option `xargs -i befehlszeile` benutzte statt des frei wählbaren Namens `ERSATZ` die feste Zeichenkette `"{}"`. Das Beispiel lautete dann:

```
find -print0 | xargs -0 -i mv "{}" "{}".bak
```

Aufpassen muss man, weil `xargs` mit der I- und der i-Option den Befehl für jede *Eingabezeile* aufruft, also ein oder mehrere Eingabeworte sammelt bis zum Zeilenumbruch.

Für die Arbeit im Dateisystem kann es aber passieren, dass sich ein Zeilenumbruch in einem Datei- oder Verzeichnisnamen befindet; daher ist hier die Option `-0` unbedingt erforderlich. Auch hier ein Beispiel:

Terminal

```
schueler@debian964:~$ ls /tmp/*.txt
fluesse.txt grepbeispiele.txt staedte.txt
schueler@debian964:~$ find /tmp -name "*.txt" \
  | xargs -I XYZ cp XYZ XYZ.bak
schueler@debian964:~$ # oder: find /tmp -name "*.txt" \
  | xargs -i cp {} {}.bak
schueler@debian964:~$ ls /tmp/*.bak
fluesse.txt.bak grepbeispiele.txt.bak staedte.txt.bak
```

1.9.9 Vergleich Kommandosubstitution – `xargs`

Vorteile der Kommandosubstitution:

- Kürzerer Aufruf
- Funktioniert auch mit eingebauten Befehlen
- Kein weiterer Prozess wird erzeugt (minimal schneller)
- `set -x` erlaubt Einsicht in den Ablauf

Vorteile von `xargs`:

- Reihenfolge der Befehle auf der Befehlszeile entspricht dem Datenfluss
- kein Problem mit zu langer Befehlszeile
- mit Option `-r` kein Problem bei fehlender Ausgabe
- mit Option `-0` kein Problem bei Leer- und Sonderzeichen in der Ausgabe
- mit Option `-n1` kein Problem bei Befehlen, die nur einen Parameter verarbeiten können
- mit Option `-I` oder `-i` kann das einzusetzende Wort auch mitten in der Zeile oder mehrmals in der Zeile stehen (siehe Beispiel)