Beispiel `05_shell2`

Dr. Günter Kolousek

14. Juli 2016

1 Allgemeines

- Wie immer: Drucke dieses Dokument **nicht** aus!
- In diesem Beispiel findet die Maus **absolut keine** Verwendung!!

2 Anleitung

1. Bis jetzt haben wir einige Bereiche schon gestreift, wie z.B. Prozesse und Rechte, sind aber nicht genauer auf die Bedeutung und Behandlung dieser Bereiche eingegangen.

   Beginnen wir mit den Prozessen. Wir wissen mittlerweile was ein Prozess ist und können auch auf einfache Art und Weise einen Prozess starten.

   Läuft ein Prozess lange, dann kann man in der aktuellen Shell keine weiteren Befehle zur Ausführung bringen, da eben der Prozess läuft und sowohl die Eingabe als auch die Ausgabe blockiert. Wie dies mit der Ein- als auch der Ausgabe funktioniert schauen wir uns im Anschluss an die Prozesse an. Jetzt stehen wir vor dem Problem, dass wir Befehle in der Shell absetzen wollen, aber dies nicht können.

   Hmm, wir könnten natürlich ein weiteres Terminal öffnen, das dann wieder eine Shell enthält und dort weiterarbeiten, aber das ist meist nicht sehr sinnvoll. Allerdings bemerken wir daran eindeutig, dass wir es mit einem multi-tasking Betriebssystem zu tun haben. Das bedeutet, dass das Betriebssystem in der Lage ist mehrere Prozesse gleichzeitig (oder quasi gleichzeitig) abzuarbeiten.

   Jetzt stellt sich die Frage, ob man nicht auch in einer Shell mehrere Prozesse gleichzeitig ablaufen lassen kann. Ja natürlich! Es soll ein Prozess einfach im Hintergrund gestartet werden, dann ist die Shell in der Lage weitere Befehle entgegen zu nehmen.

Probieren wir es einfach aus! Es gibt ein Programm `sleep`, das ein Argument erwartet, nämlich die Anzahl der Sekunden, die es schlafen soll. Probiere es einmal mit z.B. 10 Sekunden aus. Das Programm wird als Prozess im Vordergrund gestaret und belegt für die gegebene Anzahl an Sekunden auch die Eingabe.

Jetzt starten wir das gleiche Programm im Hintergrund. Dazu muss lediglich ein Ampersand hinten angehängt werden, also `sleep 10&` Ausprobieren!

2. Ok, jetzt wissen wir wie wir ein Programm im Hintergrund ablaufen lassen können. Allerdings wäre es nett, wenn wir eine Übersicht über alle laufenden Hintergrundprozesse in unserer Shell bekommen könnten. Auch dafür gibt es einen (eingebauten) Befehl: `jobs`

3. Solche "Jobs" können vorzeitig beendet werden. Nehmen wir einmal an, dass wir einen Prozess gestartet haben, der zu lange dauert. Vielleicht haben wir uns vertippt und `sleep 1000&` eingegeben. `jobs` zeigt uns auch eine Jobnummer (job ID) an, die diesen Prozess eindeutig innerhalb unserer Shell identifiziert.

   Prozesse können beendet mit dem Programm `kill` beendet werden. Dazu muss natürlich der zu beendende Prozess angegeben werden. Eine Möglichkeit dazu ist die Jobnummer, der ein Prozentzeichen vorgestellt werden muss, also z.B. `kill %1`

   Und hat es funktioniert?

4. Ja, aber es geht auch anders. Starte wieder unseren langen Prozess, aber lasse dir die Liste der Prozesse mittels `ps` anzeigen (oder mittels `jobs -l`). In dieser Liste wirst du auch dein `sleep` wieder finden. `ps` wird dir am Anfang jedes Prozesses die "process id" (pid oder PID, Prozessnummer) anzeigen. Diese identifiziert den Prozess eindeutig in deinem gesamten System und diese PID kann ebenfalls dem `kill` Befehl als Argument übergeben werden. Aber diesmal bitte *ohne* Prozentzeichen. Probiere es wieder gleich aus.

5. Gehen wir noch einmal zu dem Anwendungsfall zurück, dass ein Programm gestartet wurde und dieses sehr lange im Vordergrund läuft (also ohne Angabe des Ampersands). Wie so ein Prozess beendet werden kann haben wir schon gelernt… Wie geht das nocheinmal. Denken. Denken. Denken.

   Ok, `C-c` hilft weiter. Damit wird der Prozess allerdings beendet. Was aber wenn ich diesen Prozess nur zeitweilig stoppen will und zu einem späteren Zeitpunkt wieder fortsetzen will? Da hilft die Tastenkombination `C-z`.

   Wenn du dir jetzt die Ausgabe von `jobs` ansiehst, wirst du einen entsprechenden Unterschied sehen. Ausprobieren!

   Solche gestoppten Jobs können wieder fortgesetzt werden, indem diese mit dem Kommando `fg` (foreground) in den Vordergrund gebracht werden.

   Probiere es aus!

Was siehst du? Nichts? Sehr gut. Es handelt sich ja um das `sleep` Kommando, es hat wieder die Kontrolle über die Ein- und Ausgabe bekommen und `sleep` tut ja nicht besonders viel, nicht wahr.

Eine Information am Rande: Dem Kommando `fg` kann auch die Jobnummer mitgegeben werden, dann wird eben genau dieser Prozess in den Vordergrund gebracht. Das ist nützlich wenn man mehrere Jobs auf diese Art verwalten will. Wird keine Jobnummer angegeben (wie oben), dann der zuletzt zeitweilig gestoppte Prozess in den Vordergrund geholt.

6. Beende diesen Prozess im Vordergrund jetzt endgültig. Du weißt schon wie das geht.

   Starte danach unseren bekannten `sleep 1000` Prozess im Vordergrund. Wie wir wissen können wir diesen zeitweilig stoppen. Los geht's.

   Was aber, wenn wir den jetzt zeitweilig gestoppten Prozess im Hintergrund weiterlaufen lassen wollen? Dann kann dies mittels `bg` erreicht werden. Gleich ausprobieren!

   Wie bei `fg` kann auch bei `bg` eine Jobnummer als Argument mitgegeben werden, wenn du einen bestimmten Job im Hintergrund laufen lassen willst.

7. So, jetzt wissen wir wie dies mit den Prozessen und den Jobs funktioniert und haben uns auch den Mechanismus angesehen wie wir einen bestimmten Prozess beenden können, nämlich mit dem Kommando `kill`.

   Jetzt werden wir uns die Funktionsweise etwas genauer ansehen. Wir stellen uns die Frage wie das Kommando `kill` das so macht, dass der Prozess beendet wird?

   Es gibt das Konzept der Signale und `kill` schickt an den den betreffenden Prozess ein Signal SIGTERM. Daraufhin wird sich der Prozess dann in der Regel beenden. Warum in der Regel? Weil ein Prozess auch so einen "Wunsch" ignorieren kann.

   Was macht man allerdings mit solchen hartnäckigen Prozessen? Es gibt auch Signale, die nicht ignoriert werden können, wie z.B. SIGKILL. Mittels `kill -SIGKILL pid` (es funktioniert auch die Angabe mittels Jobnummer) wird dem Prozess das Signal SIGKILL geschickt, das er nicht ablehnen kann!

   Eine kürzere Schreibweise ist `kill -KILL pid` oder noch kürzer wäre `kill -9 pid`. Jedes Signal hat eine zugeordnete Nummer, die im Fall von SIGKILL eben 9 ist.

   Woher bekommt man diese Informationen? Mittels `man 7 signal` erhält man diese Informationen in gewohnter Weise. Die Zahl 7 gibt nur an, dass es sich um Sektion 7 handelt. Das liegt daran, dass die Manualseiten in Sektionen unterteilt sind und es mehrere Seiten mit dem gleichen Namen aber in verschiedenen Sektionen geben kann. In unserem konkreten Fall interessieren wir uns eben für die Sektion 7. Wenn du jetzt ganz neugierig bist und wissen willst welche Sektionen es so gibt, dann empfehle ich ein gepfegtes `man man`!

8. In der Manualpage `signal` der Sektion 7 findest du auch andere Signale, aber die wichtigsten sind:

   - `INT` … Interrupt, wird z.B. mittels `C-c` erreicht
   - `TERM` … Terminate, wird z.B. mittels dem normalen `kill` erreicht
   - `KILL` … Kill
   - `HUP` … Hangup, ein Relikt aus älteren Zeiten, wird oft für Serverprogramme verwendet, wenn diese ihre Konfiguration neu einlesen sollen. Für uns derzeit nicht relevant.
   - `SEGV` … Segmentation fault, wird von dem Betriebssystem an ein Prozess geschickt, wenn dieser auf einen Speicherbereich zugreifen will, der außerhalb seines eigenen Speichers liegt. Das wirst du im Laufe dieses Schuljahres noch zu Gesicht bekommen…
   - `FPE` … Floating Point Exception, z.B. bei Division durch 0 bei Gleitkommazahlen

9. Wir haben jetzt schon öfter über Ein- und Ausgabe gesprochen. Wie du aus dem Vorjahr schon weißt, handelt es sich hierbei um `stdin` und `stdout`. Jeder Prozess, den wir auf der Shell starten, hat den Eingabekanal `stdin` und den Ausgabekanal `stdout` zugeordnet. Weiters gibt es auch noch einen speziellen Ausgabekanal `stderr`, der für die Fehlermeldungen zuständig ist.

   Seitens der Shell kann man da so einiges mit diesen Kanälen anstellen. Schauen wir uns einmal das Programm `date` im Überblick an. Es gibt das aktuelle Systemdatum samt der Systemzeit auf `stdout` aus. Probiere das gleich einmal aus.

   Was aber, wenn wir diese Informationen lieber in einer Datei hätten? Dann können wir die Shell anweisen, dass `stdout` für diesen Prozess in eine Datei umgeleitet werden soll. Das kann so aussehen: `date > date.txt` Klar, dass du das auch ausprobierst.

10. Startet du diesen Befehl ein zweites Mal, dann hast du in dieser Datei die neue Zeit. Was aber, wenn du die neue Zeit ebenfalls in dieser Datei haben willst? Dann verwendest du `>>` anstatt `>`. Überprüfe, ob das stimmt.

11. Die gegenteilige Aktion zum Umleiten der Ausgabe ist das Umleiten der Eingabe. Will man, dass ein Prozess seine Eingabe nicht von der Tastatur sondern aus einer Datei liest, dann verwendet man den Operator `<` z.B. in der folgenden Weise: `wc < date.txt`

    Probiere es aus! Aha, was macht `wc`? Ziehe die Manualseite zu Rate.

12. Der Vollständigkeit halber erkläre ich dir jetzt auch wie `stderr` in eine Datei umgeleitet werden kann. Lege zuerst eine beliebige Datei `badsyntax.cpp` an und befülle diese mit beliebigen Inhalt. Will man ein `C` bzw. `C++` übersetzen, dann kann man dies folgendermaßen erledigen: `g++ -std=c++14 badsyntax.cpp` Klarerweise wirst

du Fehlermeldungen erhalten, da du ja die Datei `badsyntax.cpp` mit beliebigen Inhalt befüllt hast.

Diese Fehlermeldungen werden vom Compiler allerdings nicht nach `stdout` geschrieben sondern nach `stderr`. Will an diese Fehlermeldungen in eine Datei schreiben, dann geht dies folgendermaßen: `g++ -std=c++14 badsyntax.cpp 2> errors.txt`.

Ausprobieren!

Vielleicht stellst du dir jetzt die Frage was dieser 2er zu bedeuten hat. Das liegt daran, dass jede offene Datei unter Linux einen Dateihandle (letztendlich eine Zahl) zugeordnet und für `stderr` ist die Zahl 2 zugeordnet. Nebenbei erwähnt: `stdin` hat 0 und `stdout` hat 1 zugeordnet.

Will man `stdout` und `stderr` gleichzeitig in eine Datei umleiten, dann ist anstatt `>` bzw. `2>` der Umleitungsoperator `&>` zu verwenden.

13. Führst du das Kommando des vorhergehenden Punktes öfters auf, dann bemerkst du… Ok, gleich einmal ausprobieren.

    Also du bemerkst, dass der Inhalt sich nicht ändert, da der alte Inhalt jedes Mal überschrieben wird.

    Willst du allerdings, dass bei jedem Aufruf des Kommandos die Ausgabe hinten an die Datei angehängt wird, dann ist jeweils statt einem `>` ein doppeltes `>>` zu verwenden.

    D.h. anstatt `>` verwendest du `>>`, anstatt `2>` verwendest du `2>>` und anstatt `&>` verwendest du `&>>`.

14. Manchmal will man die Ausgabe eines Prozesses direkt mit der Eingabe eines anderen Prozesses verbinden ohne den Umweg über eine Datei zu nehmen. Das kann mit dem Pipe-Operator `|` erreicht werden: `date | wc`. Ausprobieren.

    Wieviele Prozesse hast du jetzt gestartet?

15. Wenden wir uns einem anderen wichtigen Thema zu, nämlich den Rechten. Wie wir schon gelernt haben, gehört jeder Benutzer zu einer (Haupt-)Gruppe. Diese Informationen haben wir schon bei der Ausgabe des Kommandos `id` gesehen.

    Weiters gehört jede Datei einem Benutzer und ist auch einer Gruppe zugeordnet. Diese Zuordnung haben wir schon bei der Ausgabe des Kommandos `ls -l` sehen können.

    Außerdem haben wir bei `ls -l` gesehen, dass sowohl für den Eigentümer der Datei, als auch der Gruppe als auch den Rest der Welt jeweils Rechte vergeben sind. Diese betreffen hauptsächlich das Lesen (`r`), das Schreiben (`w`) und das Auführen (`x`).

    Jetzt stellt sich die Frage wie diese beiden Informationen zusammen gefügt werden.

a) Bei jedem Zugriff auf eine Datei wird überprüft, ob der zugreifende Benutzer mit dem Eigentümer der Datei übereinstimmt. Ist dies der Fall, dann werden die Rechte der Datei bzgl. Benutzer überprüft: Hat der Benutzer das Recht auf die Datei im entsprechenden Modus zuzugreifen, dann wird es vom Betriebssystem erlaubt, anderenfalls wird der Zugriff abgebrochen.

b) Handelt es sich bei dem Zugreifer nicht um den Eigentümer, ist der Zugreifer jedoch in der Gruppe enthalten, die der Datei zugeordnet ist, dann werden die Rechte der Gruppe überprüft.

c) Ist der Zugreifer auch nicht in der Gruppe enthalten, dann werden die Rechte für alle anderen ("other") überprüft.

So, jetzt zum praktischen Teil:

a) Lege eine Datei `test.txt` in deinem aktuellen Verzeichnis an, füge dieser Datei mit dem Editor einen beliebigen Text hinzu und zeige dir den Inhalt mit `cat` an. Das funktioniert. Dann nimm dir selber (als dem Eigentümer) die Rechte zum Lesen mittels `chmod u-r test.txt` weg. Überprüfe dies einerseits mit `ls -l` und andererseits indem du versuchst mit `cat` den Inhalt auszugeben.

b) Beachte, dass du noch den Inhalt der Datei mit deinem Editor schreiben kannst. Ausprobieren!

Im nächsten Schritt geben wir uns wieder das Recht zum Lesen, nehmen aber das Recht zum Schreiben weg. Das geht mit zwei Befehlen hintereinander (hinzufügen eines Rechtes mit `+` anstatt mit einem `-` bei der Option; testen) oder auch mit einem: `chmod u+r-w test.txt`

c) Das Recht zum Ausführen ist wird mit einem `x` bezeichnet (wie eXecute) und betrifft bei einfachen Dateien nur ausführbare Programme.

d) Also, `+` bedeutet Hinzufügen, `-` bedeutet Wegnehmen und `=` bedeutet Setzen von Rechten, z.B. `chmod o=r test.txt`. Danach ist genau das Leserecht gesetzt und die anderen Rechte wurden weggenommen.

e) Alternativ kann man die Rechte auch als numerischen Wert setzen, wobei jede Rechtegruppe einer Zahl von 0 bis 7 in oktaler Schreibweise entspricht: `chmod 600 test.txt`. Was macht dieses Kommando? Ausprobieren.

f) Bei Verzeichnissen sieht die Sachlage schon anders aus. Lege dir dazu ein Verzeichnis `testing` in deinem aktuellen Verzeichnis an. `r` bedeutet hier auch, dass gelesen werden kann, aber hier bedeutet es das Auslesen des Verzeichnisinhaltes z.B. mittels `ls testing`. Ein `w` bedeutet ein Schreiben wie z.B. ein Hinzufügen einer Datei. Das `x` bedeutet ein Hineinwechseln mit z.B. `cd`. Viel Spaß beim Testen!

16. Nachdem wir auch einen ersten Einblick in die Verwaltung der Rechte bekommen haben, schauen wir uns jetzt Links an. Links sind Verweise in einem Ordner, die auf

Dateien oder Verzeichnissen (werden unter Linux auch als — spezielle — Dateien betrachtet) verweisen.

Probiere folgendes aus:

```
cd testing
ln -s ~ home
ls
ls -l
ls home
```

Betrachte auch die den ersten Buchstaben in der Ausgabe von `ls -l`, für was wird der Buchstabe wohl stehen.

Die Option `-s` bedeutet, dass es sich um einen symbolischen Link handelt. Das bedeutet, dass `home` jetzt eine eigene spezielle Datei ist, deren Inhalt in unserem Fall auf das HOME-Verzeichnis verweist. Wird diese Datei verwendet, dann weiß das Betriebssystem, dass es eigentlich auf die Datei zugreifen muss, die im Inhalt angegeben ist.

17. Schauen wir uns gleich eine weitere Variante mit Links an (wir befinden uns noch immer in `testing`):

    ```
    ln ~/test.cpp test.cpp
    ls -l
    cat test.cpp
    ```

    Woran erkennt man, dass es sich um einen Link handelt? Jedenfall nicht am ersten Buchstaben! Es handelt sich um einen sogenannten Hard-Link, der lediglich einen Eintrag im Verzeichnis darstellt. Das erkennt man an der Zahl 2, die du in der Ausgabe von `ls -l` siehst und die besagt, dass auf die angegebene Datei 2 Verzeichniseinträge verweisen.

    BTW, beim Befehl `ln` kann auch das zweite Argument weggelassen werden. Probiere einmal aus:

    ```
    rm test.cpp
    ln ~/test.cpp
    ls -l
    ```

18. Es gibt allerdings auch noch andere Shells als die bash, wie z.B. fish oder xonsh. Du kannst dir zu Hause gerne einmal die fish mittels deines Paketmanagers installieren.

    Lokal kannst du xonsh installieren und ausprobieren:

    ```
    git clone https://github.com/xonsh/xonsh.git
    python setup.py install --user
    export PATH=$PATH:~/.local/bin
    xonsh --shell-type prompt_toolkit
    ```

Der Befehl `export PATH=$PATH:~/.local/bin` kann auch in die Datei `~/.bashrc` hinten angefügt werden. Diese Datei wird immer beim Starten der bash ausgeführt. Damit erweiterst du deine Umgebungsvariable `PATH` um das Verzeichnis `.local/bin`. In dieses Verzeichnis wird `xonsh` von `python setup.py install --user` installiert.

Damit kannst du `xonsh` starten und ausprobieren.

3 Übungszweck dieses Beispiels

- Prozesse, `&`, `jobs`, `bg`, `fg`
- `kill`, `CTRL-C`, `CTRL-D`,
  - `INT`, `TERM`, `KILL`, `HUP, =SEGV`, `FPE`, `USR1`, `USR2`
- `>`, `>>`, `<`, `2>`, `&>, =2>>`, `&>>`, `|`
- Rechte, `chmod`
- symlinks, link counts
- alternative Shells