
Einige Eigenschaften der Bourne-Shell und der `bash`

1. Startup-Skripte/spezielle Dateien:

<code>~/. [bashrc_]profile</code>	von Login-Shell abgearbeitet
<code>~/.bashrc</code>	bei jedem Aufruf einer <code>bash</code> abgearbeitet
<code>~/.bash_logout</code>	von Login-Shell bei jedem Logout abgearbeitet
<code>~/.bash_history</code>	enthält History der letzten Sitzung(en)

2. Dateinamenexpansion:

- nur `bash`: zusätzlich `~+` für das aktuelle und `~-` für das vorherige Arbeitsverzeichnis
- dafür keine Shellvariablen `cwd` und `owd`

3. Shellvariablen:

- durch Zuweisung vereinbaren:

variable=wert

`unset variable`

- Zugriff auf Wortlisten stets mit geschweiften Klammern:

$\${variable}[nr]$

$\${variable}[*]$ \rightsquigarrow gesamte Wortliste

Numerierung beginnt mit 0

4. Umgebungsvariablen:

`export variable`

`export -m variable` \rightsquigarrow Entfernen nur als Umgebungsvariable

5. einige Systemvariablen:

PATH, HOME, USER, SHELL, TERM, ...

6. Ein- und Ausgabeumlenkung

- Numerierung der Kanäle: `stdin` 0
`stdout` 1
`stderr` 2
- Verwendung mit `>n` und `n<`,
> für Ausgabe- und < für Eingabeumlenkung
- `noclobber` mit `set -o noclobber` ein- und
mit `set +o noclobber` ausschalten

7. Aliasvereinbarungen:

```
alias name=definition
```

```
unalias name
```

8. Arithmetik

- keine eingebaute Shell-Arithmetik
- `expr Argument`

schreibt den Wert auf `stdout`, der sich ergibt, wenn *Argument* als arithmetischer Ausdruck interpretiert wird

```
x='expr 3 + 4'
```

9. Kontrollstrukturen

- kein `repeat`, kein (einfaches) `if`

- **for**-Schleife: `for Variable in Liste`
 `do`
 `Kommando`
 `:`
 `Kommando`
 `done`

- **while**-Schleife: `while [Bedingung]`
 `do`
 `Kommando`
 `:`
 `Kommando`
 `done`

- **if-then-else-Anweisung:**

```
if [ Bedingung ]
  then
    Kommando
    :
    Kommando
  else      # optional
    Kommando
    :
    Kommando
fi
```

- *Bedingung*
 - ist kein arithmetischer Ausdruck.
 - Gleichheit kann mit einfachem = getestet werden.
 - logisches und: *Bedingung_1 -a Bedingung_2*
 - logisches oder: *Bedingung_1 -o Bedingung_2*

Prozesse, Attribute, Umgebung

Prozess: in Ausführung befindliches Programm

Attribute: Merkmale des Prozesses, die das **System** zur Prozessverwaltung benötigt
(PID, USER, Adressraum, Zustand, ...)

s. Prozesstabelle (`ps [-e] [-f]`)

Umgebung: Befehlszähler, benutzte Daten, geöffnete Dateien, ...
Umgebungsvariablen (von den **Prozessen** benutzt.)

Systemaufrufe

- C-Prozeduren zur Bedienung der Mechanismen des Systemkerns
- z.B. zur Erzeugung, Beendigung und Synchronisation von Prozessen
- werden in Standardbibliothek von C bereitgestellt
- können von C-Programmen aufgerufen werden, nicht als UNIX-Befehle
- werden in einem geschützten Modus ausgeführt

Lebenszyklus eines Prozesses

1. Systemaufruf *fork* durch einen (existierenden) Prozess

- dupliziert den aufrufenden Prozess

Elternprozess \implies *Kindprozess* (identisch)

- Kindprozess erhält neue Prozess-ID
- Kindprozess erbt Prozessprogramm und Umgebung des Elternprozesses

2. Systemaufruf *exec*

- Kind-Prozessprogramm wird durch ein neues Programm überlagert

3. (optional:) Systemaufruf *wait*

- Elternprozess wartet (Zustand: *blockiert*), bis Kindprozess terminiert

4. Systemaufruf *exit*

- beendet aufrufenden Prozess
- Fertigmeldung an wartenden Elternprozess
- übergibt *Exit-Status* (0 .. 255) an Elternprozess

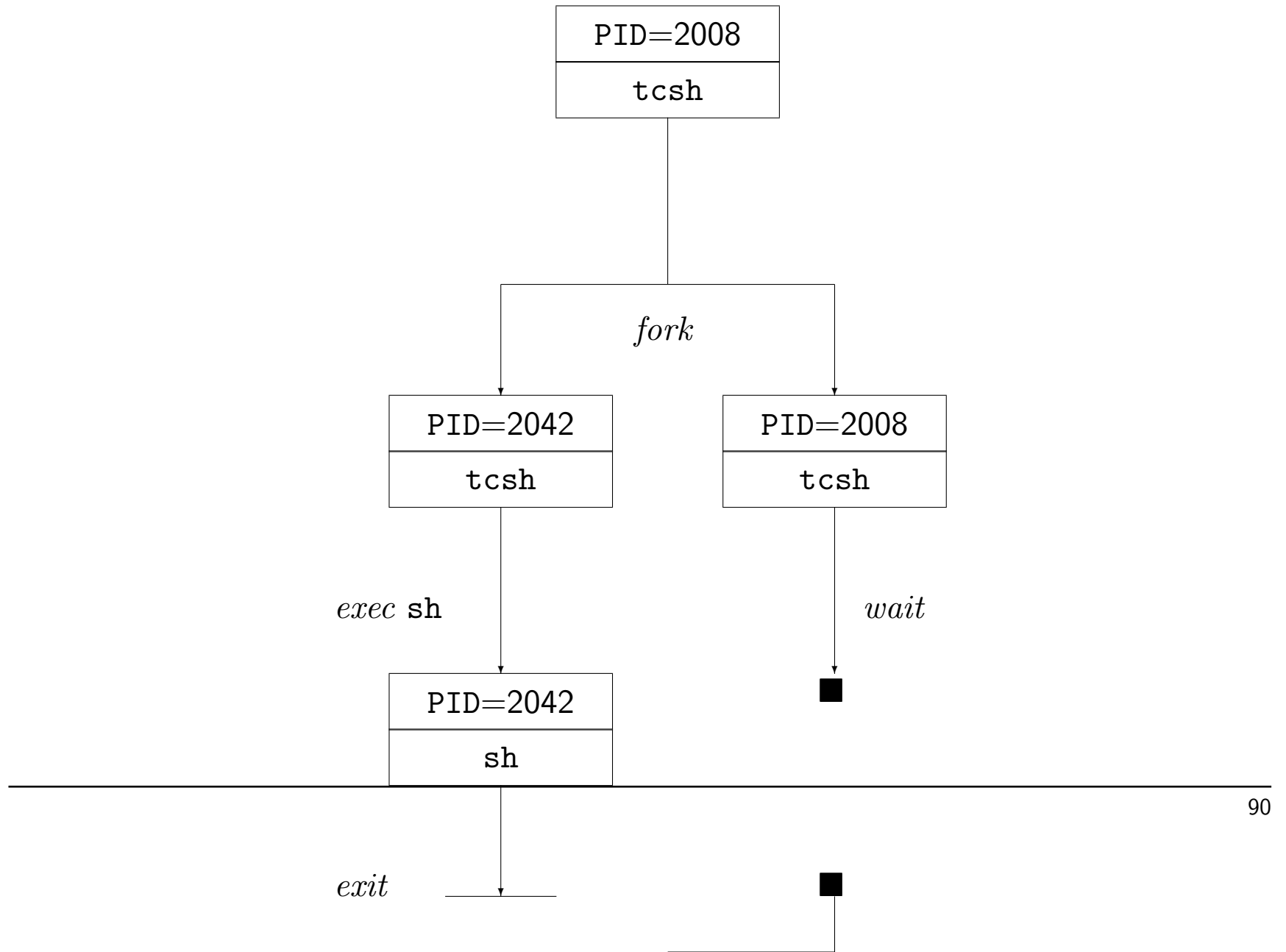
Exit-Status 0 fehlerfreies Beenden

Exit-Status > 0 Fehler (Bedeutung abhängig vom Programm)

Beispiel: Kommando `sh` in einer `tcsh`

Elternprozess: `tcsh`

Kindprozess: `sh`



Exit-Status und Verknüpfung von Kommandos

- Variable `status` speichert Exit-Status des letzten Prozesses (bash: `$?`)
- Status einer Pipeline: Status des letzten Prozesses der Pipeline
- **und**-Verknüpfung von Kommandos:
Kommando_1 && Kommando_2
↪ *Kommando_2* wird ausgeführt gdw. `status = 0`
- **oder**-Verknüpfung von Kommandos:
Kommando_1 || Kommando_2
↪ *Kommando_2* wird ausgeführt gdw. `status ≠ 0`
- Für Kommando-Verknüpfungen (`|`, `&&`, `||` und `;`) können Kommandos durch Klammern gruppiert werden (↪ Subshell-Ausführung)

Hintergrundprozesse

- Eingabe eines Kommandos \rightsquigarrow Kindprozess der Shell
 \rightsquigarrow Shell wartet auf Prozessende (*wait*)
- Hat der Kindprozess keine Ein- und Ausgabe, so ist die Shell dennoch blockiert!
- Lösung: Starten als Hintergrundprozess durch `&` hinter dem Kommando

Kommando & \rightsquigarrow [*Jobnr.*] *PID*
firefox & \rightsquigarrow [1] 28614

Jobcontrolling

- Hintergrundprozesse können über ihre *Jobnummer* angesprochen werden
- *Befehl %Jobnummer*

<i>Befehl</i>	<i>Wirkung</i>
fg	Job in den Vordergrund
bg	Job in den Hintergrund
kill	Beenden eines Hintergrundjobs
stop	Anhalten eines Hintergrundjobs
CTRL-Z	Anhalten eines Vordergrundjobs
CTRL-C	Beenden eines Jobs
jobs	Liste aller aktiven Jobs

Signale an Prozesse

- `kill [-Signalnr.] PID` (default: -15)
- Prozess mit *PID* kann auf Signle reagieren oder sie ignorieren
- Signale 9 und 23 können nicht ignoriert werden
- Beispiele:

9	unbedingtes Beenden
15	Beenden
23	unbedingtes Anhalten
24	Anhalten
25	Fortsetzen

Lokale Benutzer-Konfigurationsdateien

- `/etc/passwd`
`login:x:uid:gid:comment:home:shell`
- `/etc/shadow`
`login:passwort:lastchg:min:max:warn:inactive:expire:flag`
- `/etc/group`
`group:x:gid:user-list`
- Jeder Benutzer gehört mindestens einer Gruppe an.
Eintrag `gid` in `/etc/passwd` legt die Primärgruppe fest.
- `uid` ≥ 0 muss eindeutig im System sein; `login` ohne :

Einige der lokalen Standard-Benutzer

root	UID 0, Superuser, erhält privilegierte Shell kann ohne Passwort-Abfrage Identität jedes Benutzers annehmen Jeder Account mit UID 0 ist Superuser-Account!
daemon	UID 1, Eigentümer diverser Systemprozesse
bin	UID 2, Eigentümer von ausführbaren Dateien (Kommandos)
sys	UID 3, Eigentümer von Systemdateien

- Es existieren entsprechende Benutzergruppen.
- Nicht alle dieser Benutzer werden auf allen UNIX/LINUX-Systemen verwendet.
- Benutzerwechsel:

su [user] : Fortsetzung der Sitzung als *user* (default: root) bis exit

su - [user] : Start einer Sitzung als *user* (default: root) bis exit

Klassische Benutzerverwaltung

- Benutzer einrichten (mit Admin-Tool oder durch Kommandos:)
 1. Login-Verzeichnis (ggf. mit Startup-Skripten) anlegen
 2. Benutzer in `/etc/passwd` und `/etc/shadow` eintragen
 3. Login-Namen bei entsprechenden Gruppen in `/etc/groups` ergänzen
 4. Login-Verzeichnis mit Dateien mit `chown` an neuen Benutzer übergeben
- Benutzer entfernen: Aktionen rückgängig machen, Spool-Dateien löschen

NIS / NIS+ / LDAP

- NIS: *Network Information Service*
- LDAP: *Lightweight Directory Access Protocol*
- zentralisierte Benutzer- und Gruppenkennung im LAN – zusätzlich zu den jeweiligen lokalen Benutzern
- *Server* stellt Daten netzglobal zur Verfügung
 - Wer hat Zugang zu welchen Rechnern im Netz (*Clients*)?
 - Entsprechungen zu den Daten in `/etc/passwd` und `/etc/groups`
 - Passwörter
- *Clients* fragen bei Login-Versuchen diese Daten ab

Erweiterte Rechtehandhabung

Setzen der Rechte	Oktalwert	Name	Anzeige
<code>chmod +t datei ...</code>	1000	Sticky-Bit	t statt x im o-Block
<code>chmod g+s datei ...</code>	2000	setgid-Bit (SGID)	s statt x im g-Block
<code>chmod u+s datei ...</code>	4000	setuid-Bit (SUID)	s statt x im u-Block

- Anzeige: S bzw. T ohne x-Recht,
Anzeige: s bzw. t mit x-Recht im jeweiligen Block
- Wird eine Datei mit SUID ausgeführt, so wird der Eigentümer der Datei zum Eigentümer des Prozesses
- Wird eine Datei mit SGID ausgeführt, so wird die Benutzergruppe der Datei zur Benutzergruppe des Prozesses

- Sticky-Bit an Dateien: bei älteren UNIX-Systemen verblieben „sticky files“ im Hauptspeicher
 - ↪ schnellerer Datendurchsatz bei wiederholter Verwendung
- Sticky-Bit an Verzeichnissen: enthaltene Dateien kann nur löschen:
 - Eigentümer der Datei (falls `w`-Recht im Verzeichnis)
 - Eigentümer des Verzeichnisses
 - alle, die im Verzeichnis `w`-Recht haben, falls `w`-Recht auch für die Datei vergeben ist
 - Benutzer `root`