SV3: Linux Server

Die Shell Teil 4

Agenda

Die Shell Teil 4

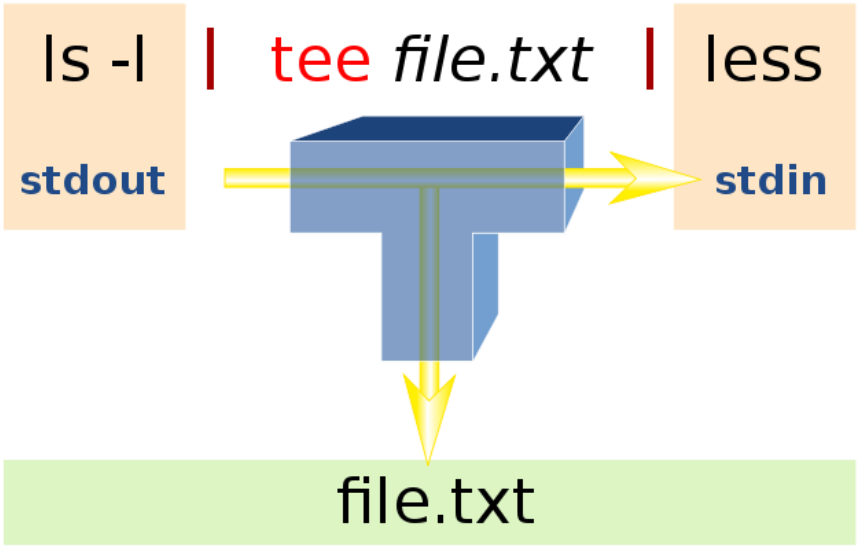
- Kommando tee
- Kommandosubstitution
- Kommando xargs
- Bedingte Kommandoausführung
- Konfigurationsdateien der Bash
- Bash-Skripting

Kommando tee

tee ist ein Standard-Unix-Kommando.

Sein Name leitet sich von dem T-Stück ab, mit dem Klempner eine Abzweigung in eine Leitung einbauen.

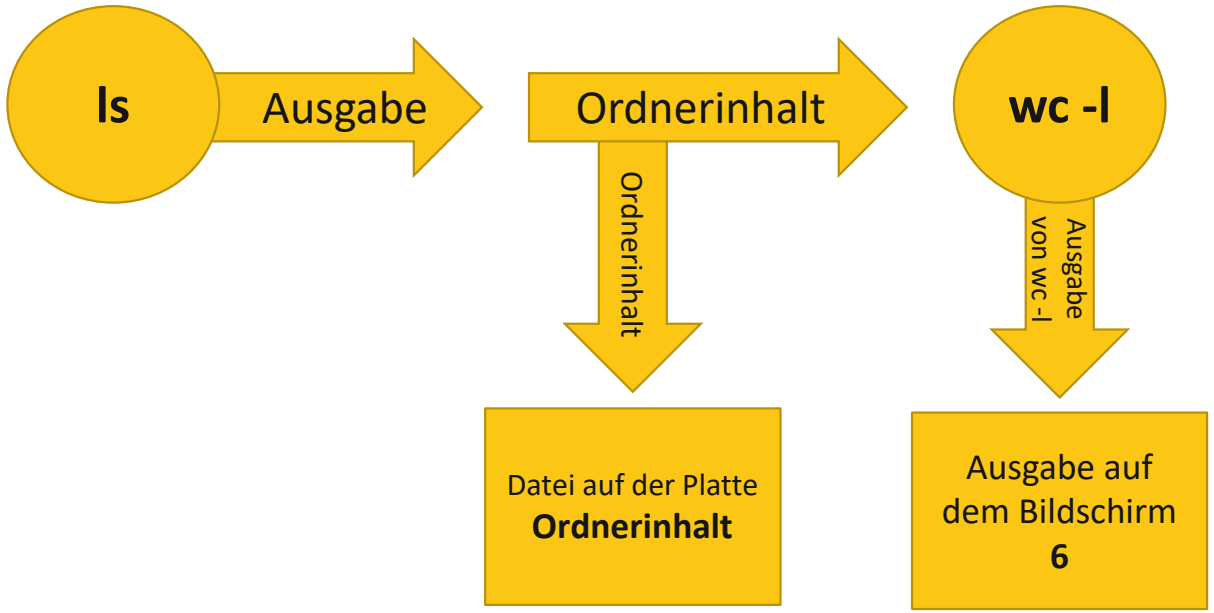
- Der Befehl liest Daten von stdin und gibt sie nach stdout und in eine Datei aus.
- Er wird benutzt, um Zwischenergebnisse innerhalb einer Pipe zu erhalten.



https://de.wikipedia.org/wiki/Tee_(Unix)#/media/File:Tee.svg

Kommando tee

tee – Kommando, um eine Pipe-Ausgabe zusätzlich in eine Datei abzuleiten



Kommando tee

Ausgabevervielfachung mit »tee«

Gelegentlich kommt es vor, dass die Ausgaben eines Programms zwar in einer Datei gespeichert werden sollen, dass Sie aber dennoch parallel am Bildschirm den Programmverlauf verfolgen wollen.

- Diese Aufgabe übernimmt das Kommando tee

```
sb@ub:~/test2$ ls | tee inhalt
1
2
3
4
inhalt
sb@ub:~/test2$ cat inhalt
1
2
3
4
inhalt
sb@ub:~/test2$
```

Kommando tee

Ausgabevervielfachung mit »tee«

Dieses Kommando zeigt die Standardausgabe auf dem Terminal an und speichert die Kopie davon in der angegebenen Datei.

- Dass es sich wirklich um eine Vervielfachung der Ausgabe handelt, bemerken Sie, wenn Sie auch die Standardausgabe von tee in eine Datei weiterleiten

```
sb@ub:~/test2$
sb@ub:~/test2$ ls | tee inhalt1 > inhalt2
sb@ub:~/test2$
sb@ub:~/test2$ cat inhalt1
1
2
3
4
inhalt1
inhalt2
sb@ub:~/test2$ cat inhalt2
1
2
3
4
inhalt1
inhalt2
```

Kommandosubstitution

Die Kommandosubstitution ermöglicht es, ein Kommando innerhalb der Kommandosubstitution durch dessen Ergebnis zu ersetzen.

- Dazu muss dieses Kommando zwischen zwei ‘-Zeichen eingeschlossen werden.
- Also z.b so **`kommando`**
- Eine alternative Schreibweise lautet **$(kommando)**

Die zweite Schreibweise ist vorzuziehen, weil sie erstens die Verwirrung durch die Verwendung von drei verschiedenen Anführungszeichen mindert (", ' und) und zweitens verschachtelt werden kann.

Kommandosubstitution

Die beiden folgenden, gleichwertigen Kommandos verdeutlichen diesen sehr leistungsfähigen Mechanismus:

```
ls -lgo `find /usr/share -name '*README*'`
ls -lgo $(find /usr/share -name '*README*')
```

Durch das obige Kommando wird zuerst find **/usr/share** -name ‚*README*' ausgeführt. Das Ergebnis dieses Kommandos ist eine Liste aller Dateien im Verzeichnis **/usr/share**, in denen die Zeichenkette README vorkommt. Diese Liste wird nun anstelle des find-Kommandos in die Kommandozeile eingesetzt. Das Ergebnis sieht so aus:

```
sb@ub:~$ ls -lgo $(find /usr/share -name '*README*')
-rw-r--r-- 1   103 Dez 19  2018 /usr/share/alsa/alsa.conf.d/README
-rw-r--r-- 1   199 Okt 10  2016 /usr/share/dict/README.select-wordlist
-rw-r--r-- 1   345 Dez 19  2012 /usr/share/doc/accountsservice/README
```

Kommandosubstitution

Dieses Ergebnis wäre durch eine einfache Pipe mit dem |-Zeichen nicht möglich.

ls erwartet keine Eingaben über die Standardeingabe und ignoriert daher auch die Informationen, die find über die Pipe liefert.

- Das folgende Kommando zeigt daher nur einfach den Inhalt des aktuellen Verzeichnisses an. Die Ergebnisse von find werden nicht angezeigt!

```
sb@ub:~$ find /usr/share -name '*README*' | ls -l # funktioniert nicht
```

Kommando xargs

Es gibt aber eine andere Lösung, die ohne Kommandosubstitution auskommt:

Durch die Zuhilfenahme des Kommandos xargs werden Daten aus der Standardeingabe an das nach xargs angegebene Kommando weitergeleitet:

```
sb@ub:~$ find /usr/share -name '*README*' | xargs ls -l
-rw-r--r-- 1 root root   103 Dez 19  2018 /usr/share/alsa/alsa.conf.d/README
-rw-r--r-- 1 root root   199 Okt 10  2016 /usr/share/dict/README.select-wordlist
-rw-r--r-- 1 root root   345 Dez 19  2012 /usr/share/doc/accountsservice/README
-rw-r--r-- 1 root root   410 Apr 21  2017 /usr/share/doc/acl/README
```

- Ein wesentlicher Vorteil von xargs besteht darin, dass es kein Größenlimit für die zu verarbeitenden Daten gibt. Die Kommandosubstitution ist hingegen durch die maximale Größe einer Kommandozeile – üblicherweise mehrere Tausend Zeichen – begrenzt.

Konfigurationsdateien der Bash

Alias-Abkürzungen

Wenn Sie das Kommando alias aufrufen, werden Ihnen die bereits vorhandenen Aliase angezeigt.

alias Alias-Name=“Kommando“

Alias ein Kommando, um eine alias-Funktion zu bilden

Mit diesem Kommando werden Abkürzungen definiert.

Konfigurationsdateien der Bash

Alias-Abkürzungen, ein Beispiel hierzu:

```
sb@ub:~$ alias suche="find . -name"
sb@ub:~$
sb@ub:~$ suche inhalt
./test/inhalt
./inhalt
./test2/inhalt
sb@ub:~$
sb@ub:~$
sb@ub:~$ alias ll="ls -la"
sb@ub:~$ ll
insgesamt 116
drwxr-xr-x 18 sb   sb   4096 Apr 16 15:53 .
drwxr-xr-x  5 root root 4096 Apr 16 13:43 ..
-rw-r--r--  1 sb   sb      0 Apr 16 15:53 1
```

Konfigurationsdateien der Bash

Alias-Abkürzungen

- Die Alias-Funktionen im Beispiel gelten allerdings nur so lange, bis man sich wieder abmeldet bzw. nur für das aktuelle Fenster, in dem man gerade arbeitet.

- Damit Sie statt ls -l immer nur ll anzugeben brauchen, schreiben Sie diese Befehle mit einem Editor in die Datei (bei der Bash) in .bashrc in Ihrem Home-Directory.

Konfigurationsdateien der Bash

Alias-Abkürzungen aufheben

Soll das kommando für die gesamte Sitzung aufgehoben werden, gibt es hierfür das Kommando:

unalias Alias-Name

unalias - das Kommando um Aliase wieder aufzuheben

Konfigurationsdateien der Bash

Alias-Abkürzungen

die wichtigsten Funktionen von alias zusammengefasst:

Befehl	Beispiel	Bedeutung
alias Alias-Name=“Befehl“	**alias** ll=“ls -l“ **alias** rm=“rm -i“	Bildung eines Alias
\alias-Name	**\rm** Datei	Aufhebung der Alias-Funktion für den aktuellen Befehl.
unalias	**unalias rm**	Alias-Funktion aufhaben
alias –x Alias-Name	**alias** –x ll=“ls -l“	Alias-Funktion auch für Subshells exportieren
alias	**alias**	Anzeige der vorhandenen Aliase.

Konfigurationsdateien der Bash

Eingabe-Prompt

Der Inhalt des Prompts wird durch die Umgebungsvariable **`PS1`** festgelegt, systemweit oft in der Datei **`/etc/bash.bashrc`**, bei Red Hat/Fedora in **`/etc/bashrc`**.

Die folgende Zeile bewirkt, dass als Prompt nur das aktuelle Verzeichnis angezeigt wird:

```
sb@ub:~$ echo $PS1
\[\e]0;\u@\h: \w\a\]${debian_chroot:+($debian_chroot)}\[\033[01;32m\]\u@\h\[\033[00m\]:\[\033[01;34m\]\w\[\033[00m\]\$
sb@ub:~$
sb@ub:~$ PS1="\w \$"
~ $ cd /tmp
/tmp $
/tmp $PS1=$PS2
sb@ub:/tmp$
```

Konfigurationsdateien der Bash

Eingabe-Prompt, dabei ist:

- \u ein Platzhalter für den Benutzernamen
- \h für den Hostnamen
- \w für das gesamte aktuelle Verzeichnis
- \W für den letzten Teil des aktuellen Verzeichnisses
- \$ für den Promptabschluss ($ oder # für root).
- \t Uhrzeit im 24-Stunden-Format HH:MM:SS

Konfigurationsdateien der Bash

Eingabe-Prompt

Farbiger Prompt Mit **\[\e[0;nnm\]** können Sie in PS1 den Formatierungs-Code nn einbetten.

- Eine umfassende Anleitung zur Prompt-Konfiguration inklusive einer Auflistung aller ANSIFarbcodes finden Sie im folgenden HOWTO-Dokument:

- http://tldp.org/HOWTO/Bash-Prompt-HOWTO

Konfigurationsdateien der Bash

Was sind die funktionalen Unterschiede zwischen .profile .bash_profile und .bashrc?

- **.bash_profile** und **.bashrc** sind spezifisch für bash
- während **.profile** von vielen Shells in Abwesenheit ihrer eigenen Shell-spezifischen Konfigurationsdateien gelesen wird. (**.profile** wurde von der ursprünglichen Bourne-Shell verwendet.)

Konfigurationsdateien der Bash

Warum das alles ?

Die Idee dahinter war, das die **`/etc/bash.bashrc`** für das gesamte System gilt. Und dann jeder User seine eigenen Einstellungen in seiner **`/home/USER/.bashrc`** vornimmt.

Abarbeitung:

Konfigurationsdateien der Bash

Konfiguration neu einlesen

Falls etwas in einer der bash Konfigurationsdateien geändert wird , müssen diese neu eingelesen werden.

source .bashrc

- Dabei sollten diese nach der richtigen Reinfolge eingelesen werden.

Variablen

Variablen haben Sie ja schon kennengelernt. Dort wurde der Wert einer Variablen zugewiesen mit:

Name=Wert; export **Name**

Optionen für Variablen:

Option	Ableitung von	Auswirkung auf den Wert der Variablen
-i	Integer	Es werden nur ganze Zahlen akzeptiert. Mit dieser Variable kann später gerechnet werden.
-u	uppercase	Zugewiesene Zeichenfolgen werden in Großbuchstaben umgewandelt. (nicht unter der Bash)
-l	lowercase	Zugewiesene Zeichenfolge werden in Kleinbuchstaben umgewandelt. (nicht unter der Bash)
-x	export	Variable wird gleich exportiert

Bash-Skripting

Wurde als Option -i integer angegeben, können Grundrechenfunktionen direkt zugewiesen werden:

Operator	Auswirkung	Beispielwert von `$erg` jeweils von Zeile zuvor	Ergebins: `echo $erg`
+	Addition	typeset –i erg=5+12	17
-	Subtraktion	erg=$erg-3	14
*	Multiplikation	erg=$erg*3	42
/	Division	erg=$erg/3	14
%	Modulo	erg=$erg%3	2

Für das Kommando `typeset -i` wird in der Korn-Shell der Alias Integer verwendet: `integer Name [=Wert]`
In der Bash verwendet man entweder `-i` oder `declare -i`

Bash-Skripting

Als Beispiel schreiben wir eine kleine Prozedur »**countdown**», die uns fragt, wie lange der Countdown laufen soll, und uns den jeweiligen aktuellen Wert des Countdowns über Bildschirm anzeigt:

```
#!/bin/bash                     # mit welcher Shell soll das Script ausgeführt werden
# Prozedur, um einen Countdown auf dem Bildschirm auszugeben
typeset -i Zahl                 # Definition der Variable als Integer
echo "Mit welcher Zahl soll der Countdown starten?"
echo "Bitte geben Sie eine Zahl ein:"
read Zahl
clear                           # clear löscht den gesamten Bildschirminhalt
echo "Der Countdown startet"
  while test $Zahl -gt 0
    do clear
    echo $Zahl; sleep 2         # sleep für Wartezeit in Sekunden
    Zahl=$Zahl-1                # Direkte Rechenoperation mit der Integer-Variable
  done
```

Bash-Skripting

Um das Script nun ausführen zu können benötigt dieses noch die Berechtigungen.

- Dafür einmal **chmod +x** DATEINAME (Über chmod später mehr)
- Danach kann das script mit: ./DATEINAME ausgeführt werden.

```
sb@ub:~/script$ chmod +x countdown.sh
sb@ub:~/script$ ls -l
insgesamt 4
-rwxr-xr-x 1 sb sb 310 Apr 17 08:26 countdown.sh
sb@ub:~/script$ ./countdown.sh
```

Bash-Skripting

In diesem Fall legt man also fest, dass für die Ausführung des Skriptes die Bash verwendet werden soll:

```
#!/bin/bash
```

```
# Dies ist ein Kommentar
kein Kommentar # Dies ist ein Kommentar.
echo "auch #kein Kommentar innerhalb" # , wohl aber außerhalb
```

Mehr zum Bash-Skripting: https://wiki.ubuntuusers.de/Shell/Bash-Skripting-Guide_f%C3%BCr_Anf%C3%A4nger/

WBS TRAINING

VIELEN DANK FÜR IHRE AUFMERKSAMKEIT!