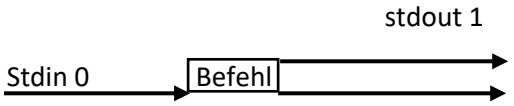
LISY (Herr Chaves)

Inhalt

Unix Kanal Konzept



Standardeingabe → stdin → Kanal 0 → Tastatur (Keyboard)

Standardausgabe → stdout → Kanal 1 → Monitor

Standardfehlerausgabe → stderr → Kanal 2 → Monitor

Kanal1 Normale Ausgabe und Kanal2 ist die Fehler Ausgabe. Kanal ist die Default Einstellung und kann mit > oder mit 1> umgeleitet werden. Errors werden mit 2> umgeleitet.

michael@VMopenSuSE:~> ls -laR /etc 2> longlist_errors_etc

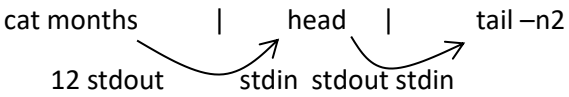
michael@VMopenSuSE:~> ls -laR /etc > longlist_etc

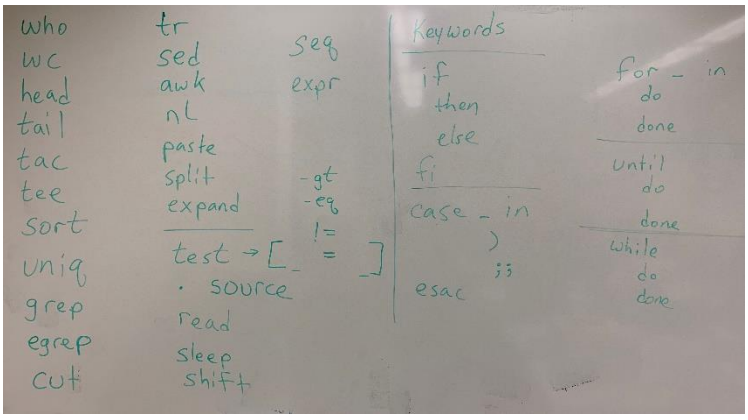
michael@VMopenSuSE:~> echo "My ps -ef command" > processes ; ps -ef >> processes

michael@VMopenSuSE:~> cat processes | less

Das „;“ trennt 2 Befehle in einer Zeile voneinander während das „|“ zwei befehle in einer Zeile vereinigt.

Der Befehl „wc -l“ ist ein Textfilter er zählt Zeilen. Er kann kombiniert mit anderen Befehlen arbeiten. z.B.: who |wc -l zeugt wieviele benutzer angemeldet sind.

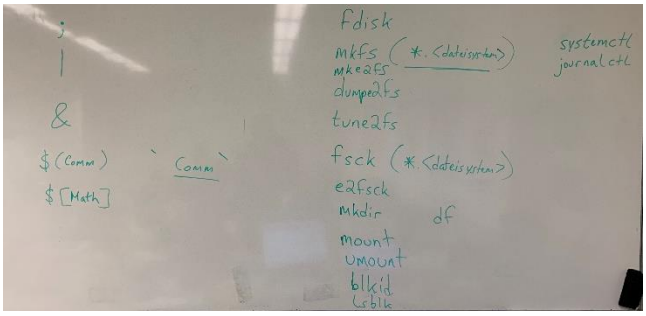




Befehle:

- wc -l
  - zählt die die lines in
- head (-n3 zeigt die ersten 3 Einträge)
  - zeigt die ersten 10 Zeilen an
- tail (-n5 zeigt die letzten 5 Einträge)
  - zeigt die letzten 10 Zeilen an
- tac
  - zeigt Text vom ende anfangend an (genau wie cat = tac)
- tee [dateiname(neu)]
  - leitet einen Daten Strom über eine Art T- stück weiter
  - zwischen einer Ausgabe von head und tail um einen zwischenstand zu erzeugen
  - z.B.: cat months|head|tee new_months|tail -n2

- sort (-k2 sortiert nach dem 2ten wort pro zeile)
  - sortiert Alphabetisch und Nummerisch
  - sort -t [Zeichen] sortiert nach einem bestimmten Zeichen (Standard ist ein blank)
- uniq
  - sortiert doppelte Wörter aus der Ausgabe aus
- grep [suchwort] [pfad]
  - Sucht multifunktionell
  - michael@VMopenSuSE:~/Dokumente> grep '^n' datafile
    - Sucht am Anfang der such nach einem „n“
  - michael@VMopenSuSE:~/Dokumente> grep '4$' datafile
    - Sucht am Ende der suche nach einer „4“
  - michael@VMopenSuSE:~/Dokumente> grep '5\..' datafile
    - Sucht nach 5, der „\“ schützt den „.“ als punkt und mit einem . in der suche innerhalb grep wird, wird ein platzhalter (ähnlich wie ? in der Bash) genutzt
  - michael@VMopenSuSE:~/Dokumente> grep -vi [suchwort] [Datei oder Pfad]
    - Sucht nach allem Außer dem Suchwort
  - michael@VMopenSuSE:~/Dokumente> grep -l 'host' /etc/* 2>/dev/null
  - michael@VMopenSuSE:~/Dokumente> grep -c 'west' datafile
  - such wieviele zeilen in der Datei enthalten sind
- egrep
  - michael@VMopenSuSE:~/Dokumente> grep -l 'host' /etc/* 2>/dev/null
  - sucht nach 2 suchmustern
  - kann auch mit „grep -E“ aufgerufen werden jedoch muss hier maskiert werden
- cut
  - schneidet Worte weg z.B.: cut -d: -f1,3,6 (-d sucht nach dem „:“ und weis somit, dass hier ein neues Wort beginnt und -f1,3,6 zeigt jedes 1, 3 und 6tes Wort an.
- tr
  - ersetzt Zeichen durch anderes Zeichen z.B.: tr : ' ' ersetzt alle „:“ durch „[LEERTASTE]“
  - tr -s '[Zeichen]' schneidet doppelte Zeichen aus und lässt nur eins übrig
- sed
  - arbeitet ähnlich wie „grep“ und „tr“
  - sed 's/Alice/John/g' alice_im_wonderland.txt „s“ (ist ein substitude (ersetzen))
- awk
  - ähnlich wie cut, schneidet raus und zeigt Felder basiert an
  - awk '{print $3, $2} ' datafile
- nl
  - schreibt zeilen- Nummerierung in Ausgabe
- paste
  - im Gegensatz zu „cut“ fügt es ein
- split
  - kann große Dateien schneiden
- test
  - test -s
  - test –f
- expand
  - kann Tabulator in Leerzeichen wandeln



Skripte erstellen

Das Skript benötigt einen Dateipfad, in dem das Skript ausgeführt werden soll (shebang- zeile).

z.B. in vim #!/bin/bash //wird in der bash ausgeführt

echo “Hello World“

Ein Script kann einmal mit der shebang- zeile definiert werden oder mit der änderung des Dateinamens auf „[Dateiname].bash“

Ausgeführt werden kann ein Skript, mit:

bash [Dateiname]

oder

.myscript // funktioniert nur wenn ich mich in diesem Verzeichniss befinde

Skripte müssen immer mit chmod so geschrieben sein, dass sie ausführbar sind(chmod u=rwx [Skript-Name]).

Verschiebt man das Skript in mein User /bin- verzeichnis, ist es von überall aus, mit eingabe des Skript- Namens ausführbar.

Ob Namen für Skripte bereits genutzt werden, kann man mit dem Befehl „type [Skript- Name]“ herausfinden.

Erstes Skript Beispiel:

```
1 #!/bin/bash
2 #
3 # This is my second try on writing a bash shell script.
4 # I hope it works!
5 #
6 clear
7 echo "Knowledge is power!"
8 echo
9 ls
10 echo
11 echo "goodbye"
```

Zweites Skript Beispiel:

```
1 #!/bin/bash
2 #
3 # This script prints user info, current date and time,
4 # and a Calender will also be printed.
5 #
6 clear
7 echo "Hello $USER"
8 echo "Today is " ; date
9 echo "Number of users logged in at the moment: "; who|wc -l
10 echo "Calender of the current Month"
11 cal
```

Skripte:

```
1 #!/bin/bash
2 #
3 # This script reads the input from a user
4 # and saves the information in a variable.
5 #
6 echo "Please enter your first name: "
7 read
8 echo "Hello $REPLY, enjoy your bash session."
```

Oder anders:

```
1 #!/bin/bash
2 #
3 read -p "Please enter your first name: " fname
4 read -p "Please enter your last name: " lname
5 echo "Hello Mr. $lname. may i call you $fname?"
```

Erklärung zu Variablen:

```
1 #!/bin/bash
2 #
3 # This script shows how to define variables
4 # and how the pre-defined system variables work.
5 # The Positional Parameters.
6 #
7 # Declaring the variables for this script
8 var1=300                          //definieren einer variablen 1 (in String)
9 var2="Hello World"                //defnieren einer variablen 2 (in String)
10 # Function test of the Positional Parameters.
11 echo "This is how many command line arguments were passed to the script: $#" //anz argumente
12 echo "This is the name of the script: $0"       //Pfad in dem das Script liegt
13 echo "This is the first argument passed to the script: $1"      //Erstes Argument
14 echo "This is the third argument passed to the script: $3"     //Drittes Argument
15 echo "All the arguments passed to this script: $*"     //Alle Argumente (nach der Skript- Eingabe)
16 echo "These are the variables that were declared in the script: $var1, $var2" //Skript- Variablen
```

Skript mit „if- Bedingung“

```
1 #!/bin/bash
2 #
3 # This script tests for the existance of a bash_history file
4 #
5 if test -f  "$1"            //kann auch if [ -f "$1" ] geschrieben werden
6   then
7     echo "The file exist!"
8   else
9     echo "The file does not exist!"
10 fi
```

Ist eine „if- Bedingung“ erfüllt, ist der Exit Status=0 und er geht in das then. Ist eine „if- Bedingung“ nicht erfüllt, ist der Exit Status ungleich 0 und er geht in else.

Shift hat die Aufgabe die vorhergehenden Umgebungsvariable zu ersetzten (heißt $1 wird immer ersetzt durch die nachstehend wartenden variablen $2, $3 usw.).

```
1 #!/bin/bash
2 #
3 # This script tests if a number is a positive number or not.
4 #
5 if [ $1 -gt 0 ]
6 then
7       echo "This number is a positive number"
8 else
9       if [ $1 -eq 0 ]
10              then
11                      echo "This is the number Zero"
12              else
13                      echo "This number is a negative number"
14              fi
15 fi
```

Skript mit „case“

```
1 #!/bin/bash
2 #
3 # This script will show how to use a case Statement
4 # using an interactive question with a user.
5 #
6 echo "What type of pet do you own (i.e. dog, cat, etc...)?"
7 read pet
8 case $pet in
9       dog)
10              echo "So you have a dog..."
11              echo "That's nice. I also have a dog."
12              ;;
13       cat)
14              echo "So you have a cat..."
15              echo "You mean, your cat owns a human."
16              ;;
```

```
17      alpacka)
18              echo "That's unusual..."
19              ;;
20      *)
21              echo "Oh... I have never heard of a $pet"
22              ;;
23 esac
```

Skript mit for- schleife:

```
 1 #!/bin/bash
 2 #
 3 # This for loop runs once for each item in the list.
 4 #
 5 for x in 1 2 3 4 5 6
 6 do
 7       echo $x
 8       sleep 2
 9 done
```

```
 1 #!/bin/bash
 2 #
 3 # This for loop runs once for each item in the list.
 4 #
 5 clear
 6 echo "**************************************************"
 7 for x in $(ls)  //oder auch (seq 1 2 20) zählt von 1-20 mit einer 2er Sprungweite
 8 do
 9       echo "This is a file in the current directory: " $x
10       sleep 1
11 done
12 echo "**************************************************"
```

Skript mit while- Schleife:

```
1 #!/bin/bash
2 #
3 # The while loop runs as long as the condition tested is true.
4 #
5 while [ "$*" != "" ]    //Liest die nach dem Befehl eingegebenen Argumente
6 do
7       echo "Argument value is: $1"
8       shift
9       sleep 1
10 done
```

Skript mit until- Schleife:

```
1 #!/bin/bash
2 #
3 # The until loop runs until the condition tested is true.
4 #
5 until [ "$1" = "" ]      //Liest die nach dem Befehl eingegebenen Argumente
6 do
7       echo "Argument value is: $1"
8       shift
9       sleep 1
10 done
```

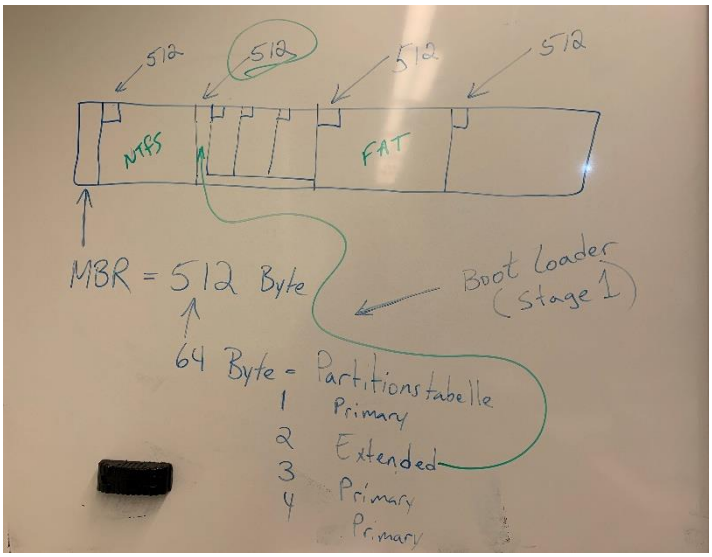
Partitionieren und Formatieren unter Linux

- fdisk -l [Pfad] //z.B.: /dev/sdb zeigt nur sdb
  - zeigt Blockgeräte an (Festplatten)
- fdisk /dev/[Datenträgername]
  - das Programm fdisk öffnet sich nun und betrachtet nur unsere gewählte HDD
- Wichtige Menü punkte in fdisk
  - m //Zeigt Menü von fdisk
  - l //Zeigt möglich Formate die wir anwenden können
  - n //Startet eine geführte Partitionierung für eine neue Partition
  - p //Zeigt bislang erstellte Partitionen, des gewählten Block- Gerätes an
  - F //Zeigt Speicher des nicht Partitionierten Bereiches an
- mkfs.ext2 /dev/sdb1
  - sdb1 wird in das Ext2- Format formatiert
- dumpe2fs /dev/sdb1 | less
  - Zeigt Formatierung Detailliert an

- mkfs.ext2 -L Data /dev/sdb1
    - Formatiert neu mit Namen (Data)
- tune2fs -L Backup /dev/sdb1
    - Bennent das gewählte Laufwerk in Backup um
- mkfs.ext2 -L Data -m1 /dev/sdb1
    - –m1 reserviert Speicherplatz für root auf 1% des Datenträgers (Default ist 5%)
- mke2fs -t ext3 /dev/sdb2
    - ist auch möglich und eigentlich das Programm auf das mkfs.ext2 zugreift
- fsck /dev/sdb2
    - Systemcheck (sollte bei getrennter Verbindung zum Betriebssys. gemacht werden)
- e2fsck -pf /dev/sdb5
    - -p (preen) reinigt und versucht auch wiederherzustellen
- mkdir -p /mountpoint/DriveBpart1 /mountpoint/DriveBpart2
    - erstellt verzeichnis mit Unterverzeichnis
- mount /dev/sdb1 /mountpoint/DriveBpart1
    - hängt Datenträger ein
- umount /dev/sdb1
    - wirft Datenträger wieder aus

In der Datei /etc/fstab können die gemounteten Datenträger manuell angelegt werden, dass sie beim Systemstart automatisch eingehängt werden. Die fstab kann auch Manuelle mit „Vim“ bearbeitet werden. Beispiel wie so eine Datei aussieht:

```
UUID=c071addb-5604-4155-9d09-5a50b313c66e /       ext4  acl,user_xattr  0  1

UUID=421b8434-31f4-4708-8318-324cdc912313  swap    swap  defaults      0  0

UUID=aa592241-ca9b-4db8-bc6f-01babe0301d8  /home   ext2  acl,user_xattr  0  2

UUID=d90e4799-9ac9-4cfe-825e-b81ad5d12f07  /arbeit  xfs   defaults      0  0

#My Automatic mounts...

/dev/sdb2 /mountpoints/DriveBpart2              ext3  defaults      0  0

LABEL=XFS /mountpoints/DriveBpart4              xfs   defaults      0  0

UUID=eb023db2-16a2-4155-8ead-fbeb8f1162d1 /mountpoints/DriveBpart1 ext2 defaults 0 0
```



- mount -a
    - mit diesem Befehl lässt sich alles was in „fstab“ steht starten (test), allerdings müssen die Partitionen dazu auch ausgehängt sein (umount).
- lsblk
    - zeigt Block- Geräte an (in Baumstruktur)
- blkid -o list
    - zeigt Block- Geräte ausführlich und übersichtlich an (Listenansicht)

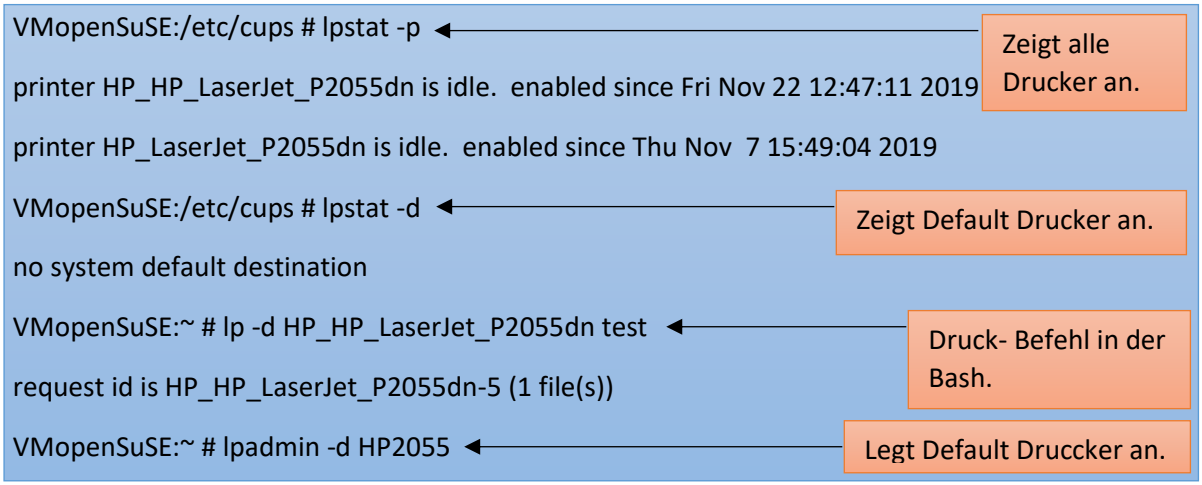
Systemverwaltung in Linux:

Mit „systemctl“ und „journalctl“ kann die Systemverwaltung bzw. eine Journaldatei anzeigen lassen. Beide zusammen sind gut für eine Fehlersuche geeignet.

- systemctl
    - zeigt alle Services die beim Start ausgeführt werden
- systemctl [start, stop, restart, reload, status, enable, disabled] [service]
    - kann starten, stoppen uvm.
- systemctl restart network
    - Startet alle Netzwerk- Services neu
- journalctl
    - Ausgabe von Systeminformationen
- journalctl -f
    - Life- Ausgabe von Systeminformationen
- journalctl -u nscd -f
    - ist ein Beispiel für eine journalsuche mit dem Schlagwort „nscd“
- journalctl --since 11:00 --until 11:54
    - zwischen einem gewählten Zeitraum

Druckerdienst „cups“

Cups ist der Lokale Druckerdienst unter Linux, den ich mit http://localhost:631 erreichen kann. Auf der Konfigurations- Oberfläche dieser Seite sind verschiedenste Einstellungen möglich.



GRUB2 Bootloader

Die Bootloder- Konfiguration kann Grafisch oder über:

VMopenSuSE:~ # cd /etc/default

VMopenSuSE:/etc/default # vi grub

Getätigt werden. Nach einer Manuellen Änderung (also ohne Grafische Oberfläche) ist es nötig

grub2-mkconfig -o /boot/grub2/grub.cfg

auszuführen, um die gemachten Änderungen wirksam zu machen.

VMopenSuSE:/etc # ls -la /etc/grub.d

Zeigt Skripte die für die Erstellung einer neuen grub.cfg benötigt werden.

Cron Job (Dienst)

Der „Cron Job“ erlaubt uns Programme oder Dienste (Skripte) Zeitlich zu steuern. C

- crontab -l
  - Listet alle Cron Job´s auf
- crontab -e
  - Legt neunen Cron Job an
- crontab -r
  - Löscht alle Cron Job´s

Zeitsteuerung :

Feld Minute	Feld Stunde	Feld Tag	Feld Monat	Feld Tag Woche
0-59	0-23	1-31	1-12	0-7
0-59/10	*	2-9	1,3,4	*

Benutzerverwaltung:

In den Dateien,

- /etc/passwd //hier sind User Infos gespeichert (kein pw, dieses ist verlinkt auf shadow)
- /etc/group
- /etc/shadow //hier sind alle Passwörter verschlüsselt abgelegt
- /etc/gshadow (falls vorhanden)

sind Dateien die, für die Benutzerverwaltung nützlich sein können.

Mit VIM ist es möglich über den Befehl „vipw“ kann die /etc/passwd auch (schnellzugriff) geöffnet werden. Durch „vipw“ ist es auch unmöglich, dass mehrere Administratoren gleichzeitig, die Datei Konfiguriert. „vipw“ ist außerdem auch in der Lage mit den Optionen:

- chfn kann weitere Informationen an einen Benutzer anhängen
- finger Benutzerinformationen
- vipw es wird die Datenbank „passwd“ editiert
- vipw -s greift auf die shadow Datei zu
- usermod -c commentzeile ändern (Name, Zimmer, Telefon)

- usermod -L sperrt einen Benutzer ohne ihn zu löschen (setzt „!“ vor das PW in Shadow)
- passwd -l Macht dasselbe wie usermod -L
- usermod -U entsperrt einen gesperrten Benutzer
- passwd -u macht dasselbe wie usermod -U
- usermod -l ändert den Usernamen (Home- Directory bleibt jedoch auf altem User)
- usermod -md ändert die Home- Directory eines Users
- chown [username:gruppe] [directory die geändert werden soll] ändert Rechte von Dateien
- chage -l [user] listet Passwort- Einstellungen auf
- chage [optionen] ändert Passwort- Einstellungen
- userdel -r löscht User zusammen mit Directory und crontab´s
- useradd -D zeigt Default Einstellungen für neue User
    - Diese Default- Daten können in /etc/default/useradd geändert werden
- visudo erlaubt es Administratorenrecht zu vergeben

```
root ALL=(ALL) ALL

michael ALL=(ALL) ALL

%admins ALL=(ALL) ALL

%ntadmin server1=(ftpuser) /usr/bin/useradd

## Uncomment to allow members of group wheel to execute any command

# %wheel ALL=(ALL) ALL
```

Einige Beispiele wie die schreibweise für Gruppen und User aussieht, innerhalb „visudo“.

Konfigurationsdateien für die Benutzerverwaltung sind:

- /etc/profile
    - Diese Datei gilt Global für jeden User der sich anmeldet
- /etc/bash.bashrc
    - Diese Datei gilt Global für jeden User der sich anmeldet
- /home/./.profile
    - Diese Datei ist versteckt (durch den . erkennbar) und gilt für den Benutzer, dessen .profile- Datei geändert wird.
- /home/./.bashrc
    - Diese Datei ist versteckt (durch den . erkennbar) und gilt für den Benutzer, dessen .bashrc- Datei geändert wird.

Repositories

Sie befinden sich in:

/etc/zypp/repos.d

In diesem Verzeichnis sind alle installierten Repositories gespeichert. Hier können auch manuell neue Repositories hinzugefügt werden. Mit touch kann eine neue Datei angelegt werden und vi geöffnet werden. In vi ist es möglich, über :r [pfad einer repo] eine schon bestehende Repositorie einzulesen (:r steht für read) und diese dann so zu bearbeiten, um eine andere einzutragen.

Der Befehl rpm -qa kann auflisten, welche software installiert ist.

rpm ist der Befehl der unterhalb zypper arbeitet, er kann also auch installieren und deinstallieren, allerdings erkennt rpm keine Programmabhängigkeiten, zypper schon.

Zypper search --provides --match-exact

Kann Programm- Abhängigkeiten in Paketen finden (beispiel: glibc beinhaltet viele kleine Pakete, diese werden mit dem Befehl durchsucht)

Tar Befehl

Mit dem tar- Befehl lassen sich archive erstellen lesen oder extrahieren.

michael@VMopenSuSE:~/Downloads> tar -cvf etc.tar /etc

erstellt (create) vorgangsanzeige (verbose) Datei erstellen neuer name was archiviert

michael@VMopenSuSE:~/Downloads> tar -cvzf etc.tar.gz /etc //packt mit gzip (z)

michael@VMopenSuSE:~/Downloads> tar -cvjf etc.tar.gz /etc //packt mit bzip2 (j)

michael@VMopenSuSE:~/Downloads> tar -tf etc.tar //kann archive anzeigen

michael@VMopenSuSE:~/Downloads> tar -xf etc.tar //entpackt normal

michael@VMopenSuSE:~/Downloads> tar -xzf etc.tar.gz //entpackt gzip archiv

Tarballs

Nach dem entpacken eines Installationsverzeichnis muss die „.configure“ Datei ausgeführt werden, danach wird eine ausführbare „Make“ Datei erzeugt, diese Datei muss mit dem Befehl make Kompiliert werden und ist danach (vorausgesetzt es sind alle Programmabhängigkeiten installiert) mit „make install“, installierbar. Nach dieser Reihe von befehlen kann das installierte Programm mit dem Namen des Programms geöffnet werden.

1. programm.tar.gz entpacken mit tar -xf programm.tar.gz
2. Readme Datei lesen mit less readme
3. configure Datei ausführen (script) mit .configure
4. In das Verzeichnis wechseln Kompilieren mit make
5. Installieren mit make install

Installationsstandards

Open SuSE / Red Hat / Fedora	Debian / Ubuntu
rpm (Softwarepaket)	dpkg (sofwarepaket)
zypper (Front- end für rpm)	apt (apt-get, apt-cache , apt-cdrom (früheres Front- end)
yum (Front- end für rpm)	apt (neues Front- end)
dnf (Front- end für rpm)	

Mögliche Prüfungsinhalte:

- Textfilter (tac cat less more tail head sort wc)
- std in std out (>> >)
- Scripte wie ausführen (als shell oder ausführbar Bit mit chmod 755 !bin/bash)
- Script erstellen schleife 1 hoch zählen
- Positions paramter ($1 $2 $3 $# $*)
- Kontrollstrukturen (if else do done) (befehle echo read shift)
- mbr gpt partitionierung
- fdisk mount ls block umount fsck
- fstab
- Filesystem ext2, ext3, ext4, ntfs, fat32,
- systemctl, systemd, journalctl,
- Cups Drucker lp lpr (druckerserver)
- Boot Prozesse grub2 mit etc/default/grub konfig ändern
- crontab erstellen listen löschen Struktur von cron selbst (crontab –u benutzer)
- Benutzer Verwaltung (useradd …, groupmod…, sudo, visudo)
- Software Verwaltung (tar, tarballs, rpm)