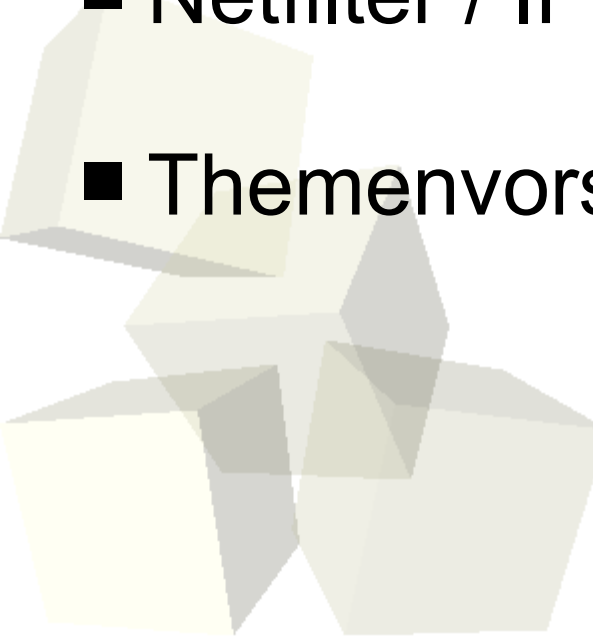


- SSH für Fortgeschrittene
- Erweitertes Rechtemanagement
- Erweitertes Paketmanagement
- Kernel backen
- Sysadmintools
- Wichtige Dienste
- Shell Programmierung
- Netfilter / IP-Tables

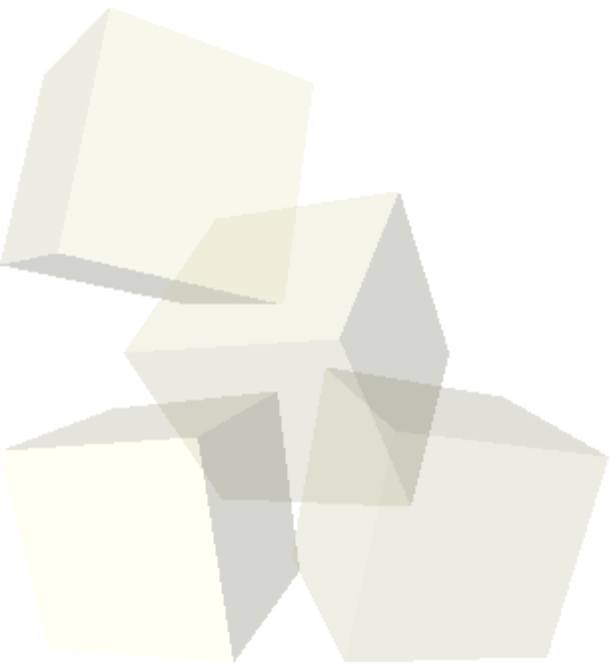
- Themenvorschläge?





Warum nicht als *root* arbeiten?

- Bedienungsfehler
- kein Logging
- Sicherheitskritisch
 - ◆ z.B.: bugs in *tar* führen dazu, dass nicht in richtigen Pfad entpackt wird, wodurch beliebige Dateien überschrieben werden können
 - ◆ Ausführung unbekannter Programme





- führt Befehle mit root-Rechten aus
- Ubuntu nutzt dies sehr konsequent, indem man normal nicht als *root* arbeiten kann
 - ◆ *root* hat kein Passwort, damit ist keine Anmeldung als *root* möglich
- *sudo* loggt alle Aktivitäten in */var/log/auth.log*
 - ◆ bei anderen Distributionen evtl. anders
- Nachvollziehbarkeit bei mehreren Admins
- User braucht **kein** root-Passwort
- Definition der Rechte in */etc/sudoers*
 - ◆ bearbeiten mit *visudo* (hat Syntaxcheck)
 - ◆ normaler Editor tuts auch



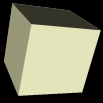
- Ubuntu: alle Mitglieder der Gruppe *admin* bekommen alle Rechte mittels sudo
- /etc/sudoers

```
root ALL=(ALL) ALL
thorben ALL=(ALL) ALL
karl ALL=(ALL) NOPASSWD :/etc/userscripts/create_email.sh,/bin/passwd
katrin server1=(ALL) /sbin/cfdisk
```

```
Cmnd_Alias SHUTDOWN = /sbin/shutdown
skx ALL = SHUTDOWN
```

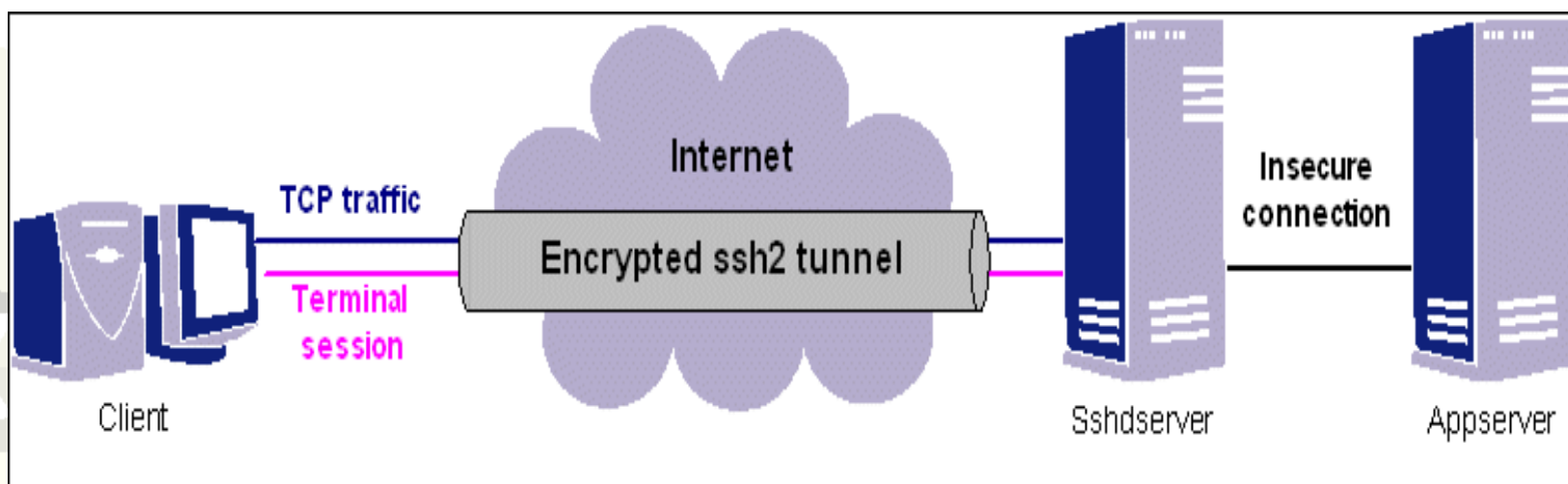
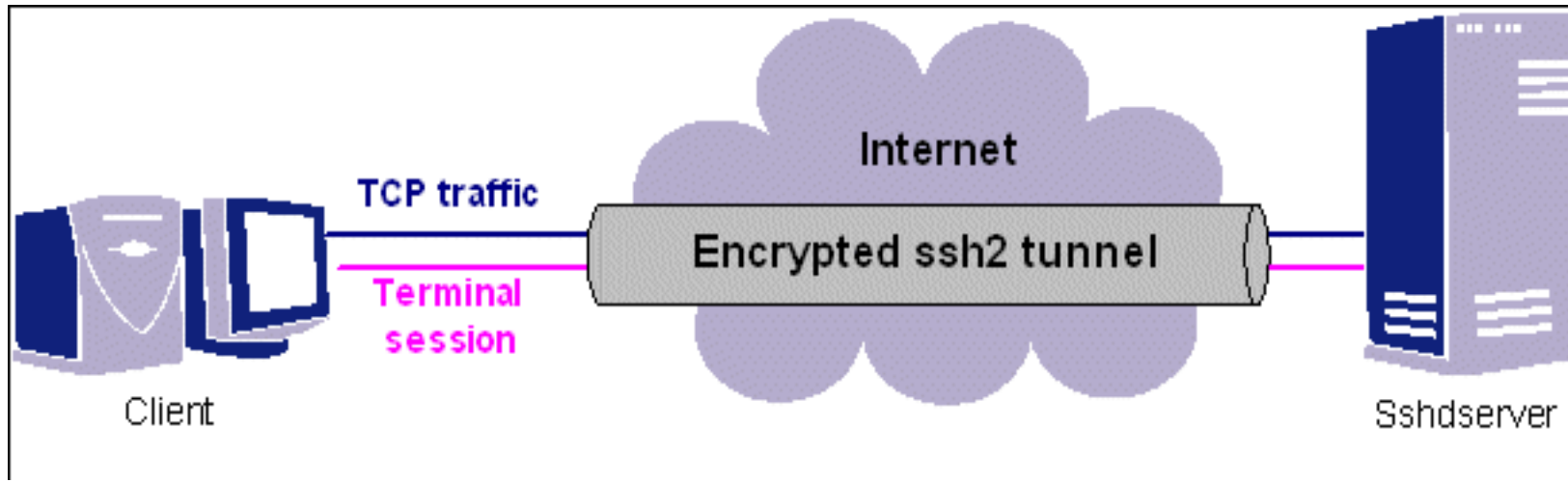
- man sudoers
 - ♦ sehr ausführliche Anleitung
- <http://www.debian-administration.org/articles/33>

- `su` = switch user
- lässt normalen Benutzer zu *root* werden
- *root* kann als beliebiger User agieren
 - ◆ `su thorben`
 - ◆ User muss gültige Shell haben
- `su` kann jeder Benutzer ausführen
- Benutzer muss root-Passwort kennen
- mit „`su -`“ werden Umgebungsvariablen des Users übernommen, in den man sich wandelt
- kein Logging der Aktivitäten
- Befehle als anderer User ausführen
 - `su -c "id" thorben`
 - `uid=1003(thorben) gid=1003(thorben) groups=1002(thorben1),1003(thorben)`



Weitere SSH Features

- SSH kann beliebige TCP-Ports weiter leiten
- Anwendungsgebiete
 - ◆ Verschlüsselung von Klartextprotokollen
 - ◆ Umgehung von Restriktionen
 - ◆ „VPN für Arme“
- `ssh -L localport:localhost:remoteport user@remotehost`
 - ◆ öffnet an lokalem Rechner auf Localport eine Verbindung zum Remoteport auf Remotehost
- `ssh -R remoteport:localhost:localport user@remotehost`
 - ◆ öffnet an remote Rechner eine Verbindung zum Localport unseres Rechners



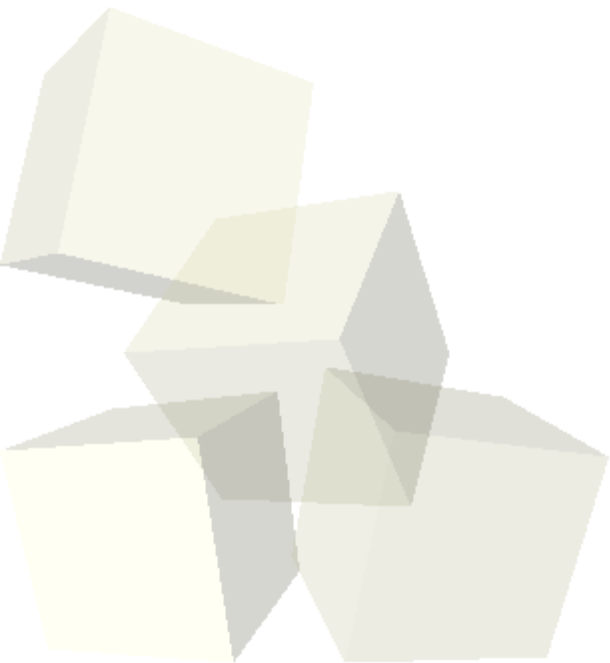


Weitere SSH Features

- per default ist geöffneter Tunnel nur lokal nutzbar
- Parameter „-g“ stellt Port an allen Interfaces bereit
 - ♦ `ssh -g -L9999:localhost:3306 thorben@paranoit.org`
- Port-Tunneling muss bei Firewallkonfiguration berücksichtigt werden
- zusätzliche Konfiguration in `.ssh/authorized_keys`
 - ♦ `command="/path/command",`
`from="ip-address",`
`no-port-forwarding,`
 - ♦ <http://www.openbsd.org/cgi-bin/man.cgi?query=sshd>
- ab Version 4.3 kann OpenSSH vollwertige VPN-Verbindungen



uebung_01_ssh fortgeschritten.pdf



Erweitertes Rechtemangement

■ Sticky Bit - 1

- ◆ Verzeichnis: nur Besitzer einer Datei darf sie löschen
→ drwxrwxrwt 5 root root 4096 Sep 24 21:05 tmp

■ SGID Bit - 2

- ◆ Verzeichnis: alle neu erstellten Dateien gehören nicht der Gruppe des Erstellers, sondern der Gruppe, die das Verzeichnis besitzt

■ SUID Bit - 4

- ◆ Dateien: Prozess läuft unter Identität der besitzenden Gruppe

→ rwsr-xr-x 1 root root 27132 /usr/bin/passwd

■ „chmod 1755 verzeichnis“

■ umask

- ♦ default Rechte einer Datei sind 666
- ♦ default Rechte eines Verzeichnisses sind 777
- ♦ *thorben@karl:~\$ umask*
0022
- ♦ */etc/profile*
 - *umask 022*
- ♦ durch Subtraktion des umask Wertes von den default Rechten ergibt sich für Dateien 644 und Verzeichnisse 755

■ Hinweis: Änderungen an */etc/profile* können mit „*source /etc/profile*“ übernommen werden



Erweiteres Rechtenmanagement

- „lsattr“ zeigt zusätzlich Attribute
- Immutable Bit
 - ◆ Datei wird unveränderbar (auch für *root*!)
 - ◆ „chattr +i datei“
 - ◆ *BSD kennen hier mehr und bessere Optionen
 - Z.B. Datei darf nur größer werden (gut für Logfiles)
 - Änderungen mit „chattr“ auch für *root* nicht möglich wenn `kern.securitylevel` hoch genug
 - ◆ schützt effektiv vor übereifrigem *root* user ;-)

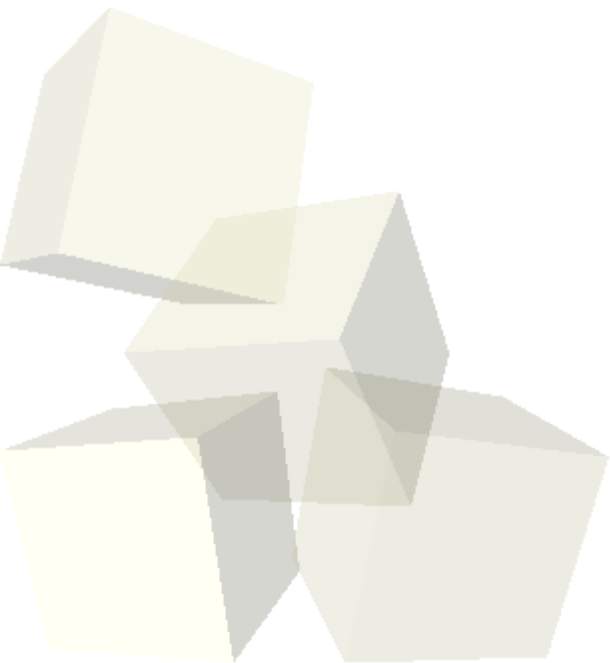


Benutzerrestriktionen / Limits

- `/etc/security/limits.conf`
 - ◆ Datei ist selbsterklärend
 - ◆ Zuordnung einer Ressource zu Benutzer
- Einschränkung von:
 - ◆ max. geöffnete Dateien
 - ◆ max. CPU Zeit
 - ◆ max. Anzahl an Prozessen
 - ◆ max. gleichzeitige Logins
 - ◆ standard Priorität mit der Prozesse ausgeführt werden
 - ◆ ...
- anzeigen mit „`ulimit -a`“ in der bash
 - ◆ zsh und tcsh bieten intern den „`limit`“



- Priorität für CPU Zeit
- -20 bis 19
- je weniger "nice" desto höher die priorität
- Anzeige mit „top“
- `nice --10 vi /etc/apt/sources.list`
- `nice -n -10 vi /etc/apt/sources.list`





- 1. Gebot: vergehe dich niemals an deinem Paketmanager!
- Paketmanager sorgt für Ordnung
 - ◆ kennt jede Datei im System, wo sie her kommt
- Paketmanager löst Abhängigkeiten bei Installationen
- Paketmanager bietet Suchfunktion, wenn man nicht genau weiß was man installieren will
- ermöglicht Administrator effizientes arbeiten
- Security: gpg-Signaturen garantieren Integrität
- Debianderivate: „apt-get“ ist veraltet, „aptitude“ seit 2 Jahren state of the art!



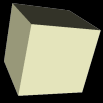
Paketmanagement - aptitude

- aptitude
 - ◆ install (installiert)
 - ◆ remove (Konfigurationsdateien bleiben erhalten)
 - ◆ purge (deinstalliert auch Konfigurationsdateien)
 - ◆ hold (Paket wird nicht aktualisiert bei Updates)
 - ◆ unhold (Paket kann wird bei Updates aktualisiert)
 - ◆ clean (löscht Paketcache)
 - /var/cache/apt/archives/
 - ◆ show (zeigt infos über Abhängigkeiten usw.)
 - ◆ search (sucht Pakete)

- bessere Suchergebnisse mit: apt-cache search
- Dateisuche: apt-file search
 - ◆ nicht standardmäßig installiert
 - ◆ vorher „apt-file update“ aufrufen
- <http://packages.ubuntu.com> - Suchfunktion



- `dpkg -l` : listet alle installierten Pakete
 - ◆ spalten oft nicht breit genug:
 - „`COLUMNS=150 dpkg -l`“
- `dpkg -L paketname` : zeigt Dateien eines Pakets
- `dpkg -S` : zeigt an aus welchem Paket Datei ist
 - ◆ `thorben@karl:~$ dpkg -S /etc/apache2/apache2.conf`
`apache2-common: /etc/apache2/apache2.conf`
- `dpkg -i paketdatei.deb` : installiert Paket
- `dpkg -r paketname` : deinstalliert Paket (besser aptitude nutzen!)
- `dpkg --purge paketname` : löscht Paket inkl. Konfigurationsdateien
- `aptitude -oDPkg::Options::="--force-confmiss"`
`reinstall PAKETNAME` : neu inkl. configs

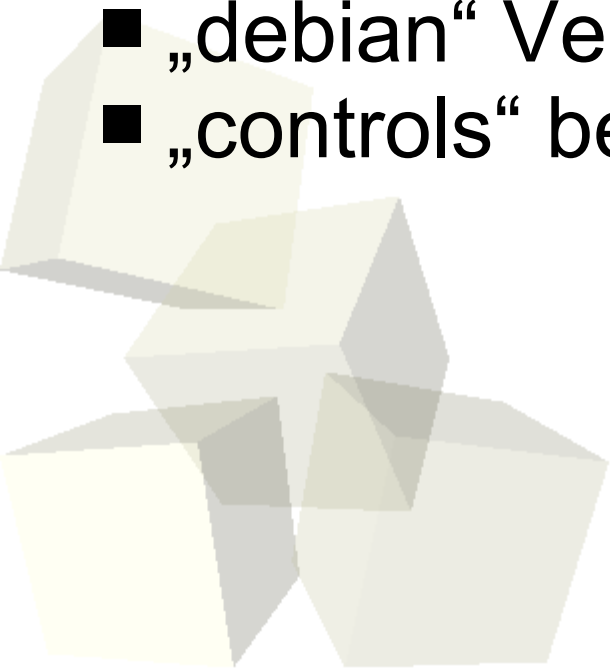


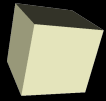
- Klassiker in HowTOs und Fachzeitschriften
 - ◆ configure, make, make install
- sorgt auf Dauer für Chaos
- installierte Dateien nur mit Hilfe von Tools auffindbar
- Installationspfad beim „configure“ angeben ist nicht komfortabel
- Lösung: Pakete selbst bauen
 - ◆ teilweise nötig, wenn Pakete neu kompiliert werden müssen wegen spezieller Einstellungen





- Vorgehensweise Paketbau
 - ◆ Quellcode Archiv entpacken
 - ◆ Entpacktes Verzeichnis muss Versionsnummer enthalten
 - ◆ „configure“ und „make“ zum testen ausführen
 - ◆ „dh_make“ ausführen
 - ◆ „dpkg-buildpackage“ ausführen
 - ◆ fertiges Paket mit „dpkg -i“ installieren
- „debian“ Verzeichnis mit Einstellmöglichkeiten
- „controls“ beinhaltet Abhängigkeiten





Paketmanagement Ergänzung

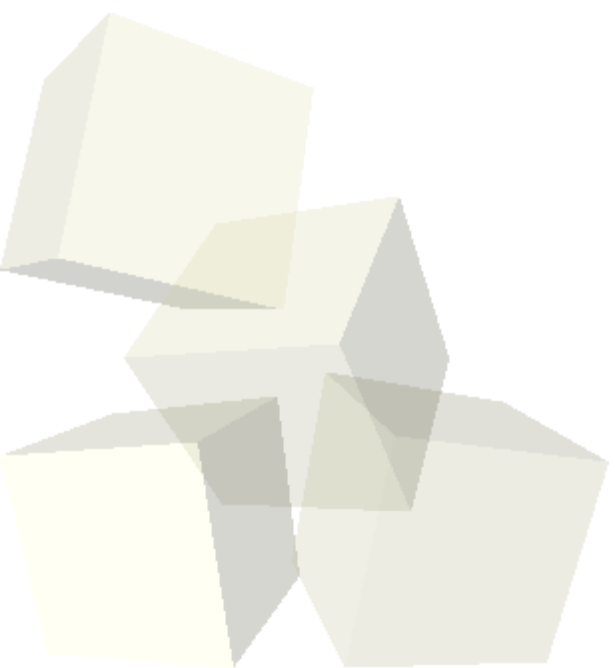
- einfacher mit „checkinstall“
 - ◆ checkinstall ab lenny in Debian
 - ◆ <http://www.asic-linux.com.mx/~izto/checkinstall/>
 - lauffähiges .deb für etch
- checkinstall kann auch .rpm erstellen
- sehr gute Dokumentation!

- configure and make wie gewohnt
- „checkinstall -D“ erstellt Debian Paket

- fertig :-)



uebung_02_paket_bauen.pdf

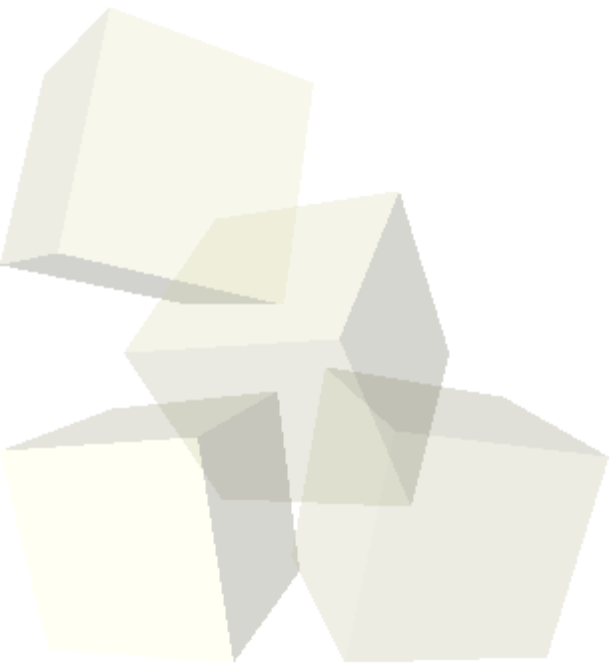


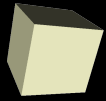


Was ist mit rpm?

- rpm Format in RedHat basierenden Distributionen
- ähnliche Features wie Debians .deb
- rpm to deb Konvertierung mit „alien“

- ich kenne mich damit nicht aus :-)





■ /var/log

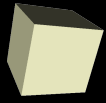
- ◆ messages generelles Log, erste Anlaufstelle
- ◆ syslog Systemlog, sollte aufgeräumt sein
- ◆ auth.log Authentisierung
- ◆ mail.log Mail
- ◆ apache / samba / mysql usw. :
 - Logdateien der Dienste

■ *tail -f logfile*

- ◆ Logfile live mitlesen
- auf Größe der Logfiles achten, sollten rotiert werden (z.B. durch „logrotate“)



- Standarddienst auf meisten *nix Systemen
- syslog-ng als Nachfolger
- Alternativen bieten Verschlüsselung, SQL...:
 - ◆ rsyslog
 - ◆ Modular Syslog
- /etc/syslog.conf
 - ◆ relativ selbsterkändig
 - ◆ Facility.Level Action
 - ◆ „-“ Zeichen am Anfang steht für asynchrones Schreiben auf die Festplatte und trägt zur Performanceverbesserung bei
- Syslog ist netzwerkfähig
 - ◆ hört auf UDP Port 514
 - ◆ zentrales Logging / Logüberwachung möglich



- Facilities (Quellen)
 - ◆ zeigt wer Log gesendet hat
- Kategorien
 - ◆ authpriv
 - ◆ cron
 - ◆ daemon
 - ◆ kern
 - ◆ lpr
 - ◆ mail
 - ◆ mark
 - ◆ news
 - ◆ syslog
 - ◆ ...



■ Severity (Log Level / Schweregrad):

- ◆ debug
- ◆ info
- ◆ notice
- ◆ warning (or warn)
- ◆ err (or error)
- ◆ crit
- ◆ alert
- ◆ emerg (or panic)

- warn, error, und panic sind veraltet, können einem aber begegnen



- Zeitgesteuertes Ausführen von Programmen
- verschiedene cron-Dienste
 - ◆ anacron, vixie-cron usw.
- Zeitpunkte der Ausführung in *crontabs*
- */etc/crontab*
 - ◆ *crontab* für das System
 - ◆ nur für *root* schreibbar
 - ◆ cron sollte Änderungen selbstständig bemerken, manchmal muss man „*crontab /etc/crontab*“ ausführen (zur Not cron restarten)
- *crontabs* für jeden Benutzer
 - ◆ */var/spool/cron/crontabs*
 - ◆ *crontab -e* editieren der eigenen *crontab*



■ Aufbau

- ♦ relativ selbsterklärend
- ♦ wann soll wer etwas tun

```
MAILTO=user@domain.org
```

```
0 4 * * * root ntpdate ntp1.fau.de ptbtime1.ptb.de
```

```
*/30 * * * * root /usr/bin/webalizer
```

■ teilweise Leerzeile am Ende nötig

■ /etc/cron*

- ♦ zur besseren Übersicht sind Aufgaben in verschiedenen Verzeichnissen zu finden

■ cron.allow / cron.deny zur Einschränkung möglich

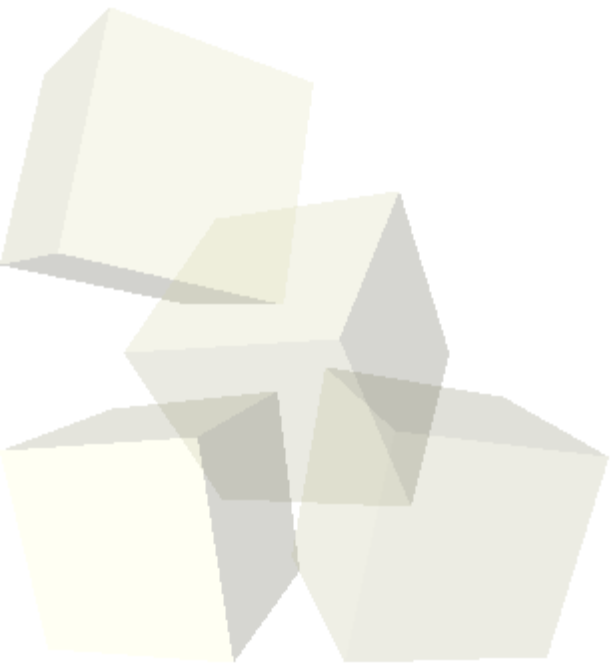
■ Beispiel:

```
# run myprogram at 6:15am and 6:15pm on the 1st and 15th
```

```
15 6,18 1,15 * * myprogram
```



- Userquotas und Groupquotas
- werden vom Dateisystem überwacht
- nicht in jedem Dateisystem verfügbar
- Hard- und Softlimits
 - ◆ Inodes oder Blöcke (Kilobyte)
- werden Partitionsweise aktiviert
 - ◆ /etc/fstab





■ /etc/fstab

- `/dev/sda7 /home ext3 defaults,usrquota,grpquota 0 0`

■ remounten der Partition

- `mount -o remount /home`

■ mit „mount“ überprüfen

- `/dev/sda7 on /home type ext3 (rw,usrquota,grpquota)`

■ `edquota <username>`

- editieren der Limits, meist mit vi

■ `repquota /mountpoint`

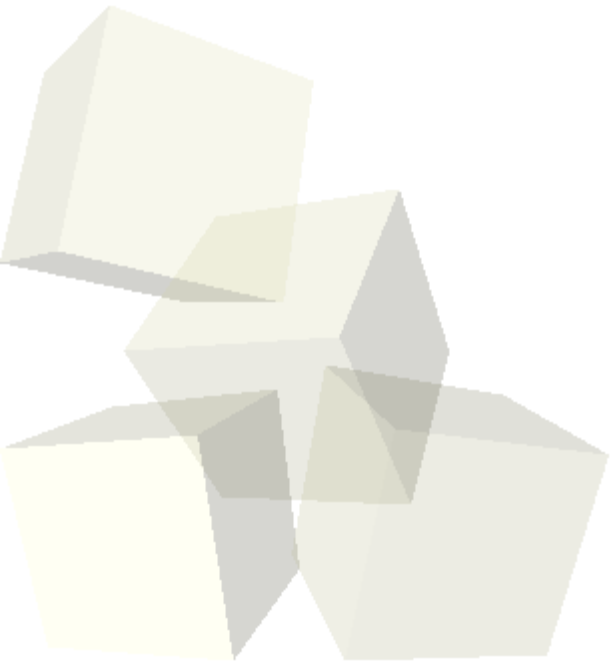
- zeigt Quotastatistiken an





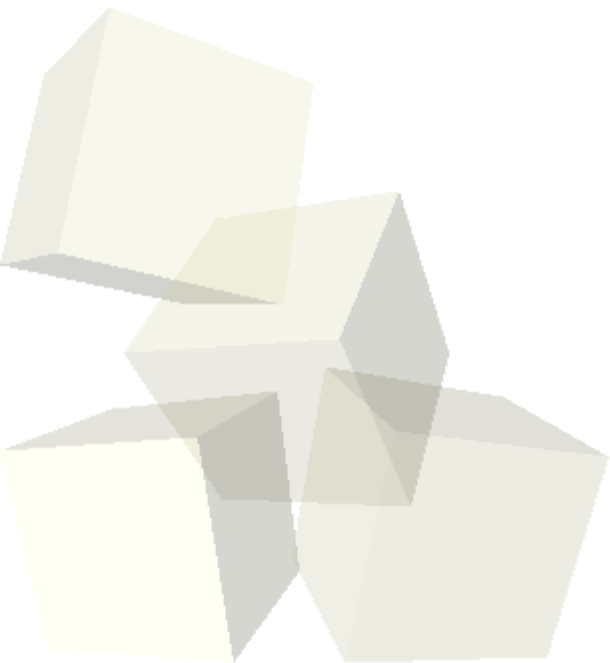
Der Bootvorgang

- Stromversorgung
- BIOS
- Bootloader – grub oder lilo
- Kernel
- Init
- Runscripts
- Shell / X-Windows





- auf Mainboard in EEPROM gespeichert
- führt Selbsttest durch
- initialisiert Hardware
- Security: Bootreihenfolge
 - ◆ Booten nur von Festplatte zulassen
 - ◆ BIOS durch Passwort schützen
- lädt Bootsektor des eingestellten Bootlaufwerks



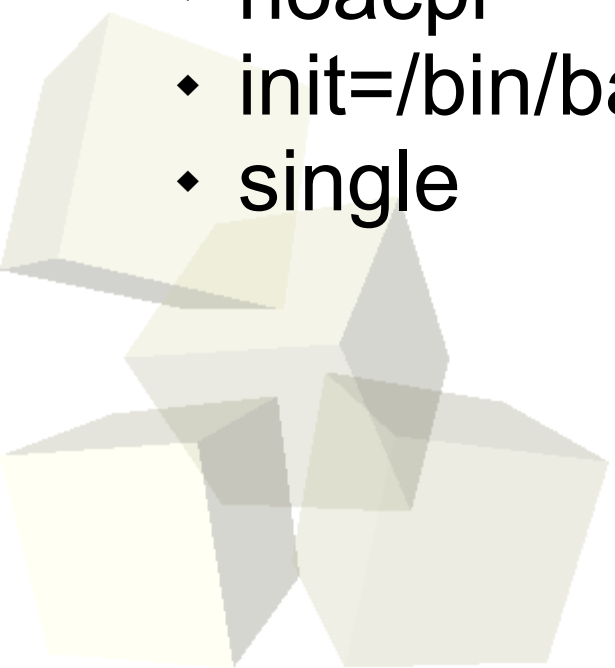


- sehr klein, da Bootsektor beschränkte Kapazität hat
- lädt den Kernel
 - ◆ kann spezielle Optionen mitgeben
 - ◆ Auswahl verschiedener Kernel
- muss Dateisystem verstehen
- heutzutage meist grub, lilo eher selten
- grub liest Informationen von Festplatte
- lilo speichert alles in MasterBootRecord
- Security: setzen eines Bootloader-Passworts um booten in den single-Modus zu verhindern



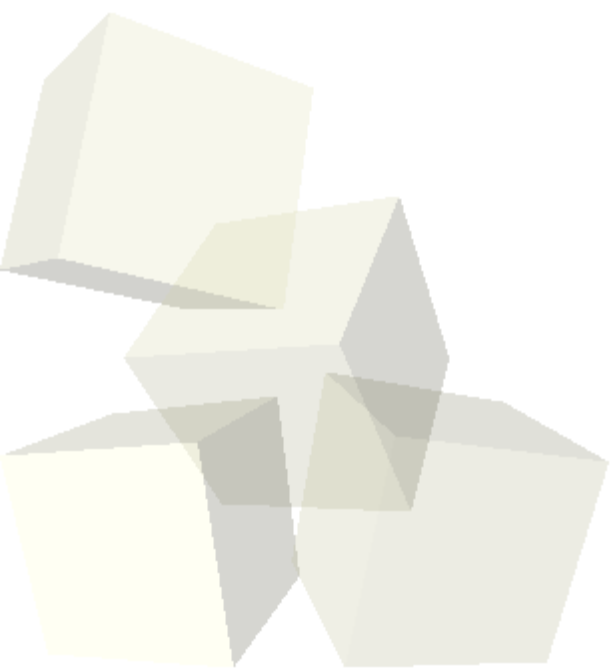
- zusätzliche Kernelparameter möglich
- `kernel /vmlinuz-2.6.9-1.667 ro root=/dev/hda2 clock=pit`

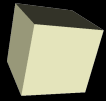
- Beispiele für Parameter
 - ◆ `clock=pit` (für Uhrzeit in VMWare)
 - ◆ `nosmp` (kein Multiprozessorsupport)
 - ◆ `noapic` (kein APIC)
 - ◆ `noacpi` (kein ACPI Powermanagement)
 - ◆ `init=/bin/bash` (alternativer INIT-Prozess)
 - ◆ `single` (in den Single-Modus booten)



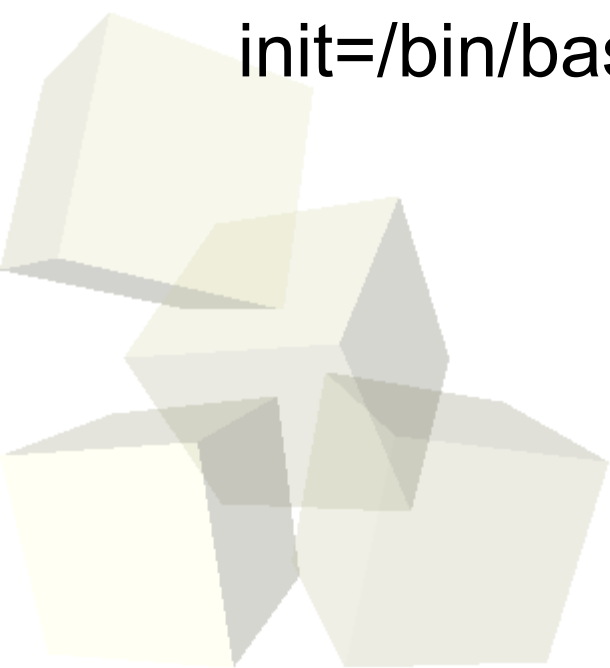


uebung_03_bootmanager.pdf



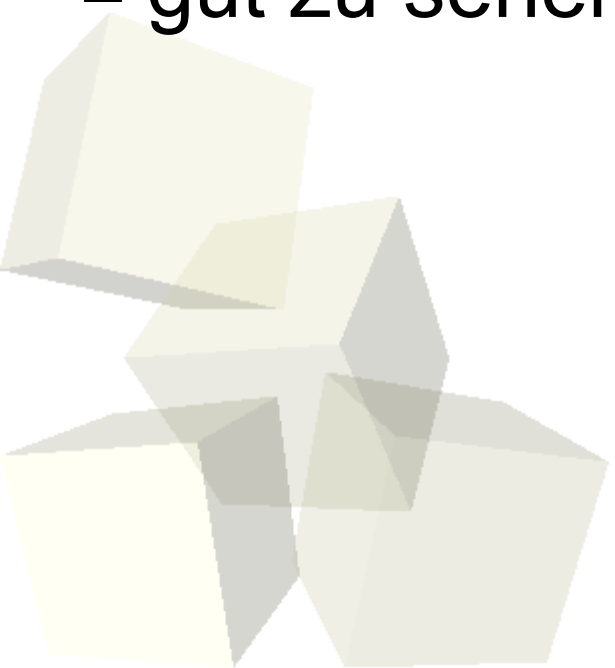


- ist gleich das Betriebssystem in Linux
- lädt Hardwaretreiber
 - ◆ erkennt vorhandene Hardware automatisch (sofern Treiber vorhanden)
 - ◆ ignoriert BIOS Einstellungen
- ruft `/sbin/init` auf
 - ◆ benötigt Angabe der `-`-Partition
 - ◆ anderer Pfad möglich mit Bootoption (z.B. `init=/bin/bash`)



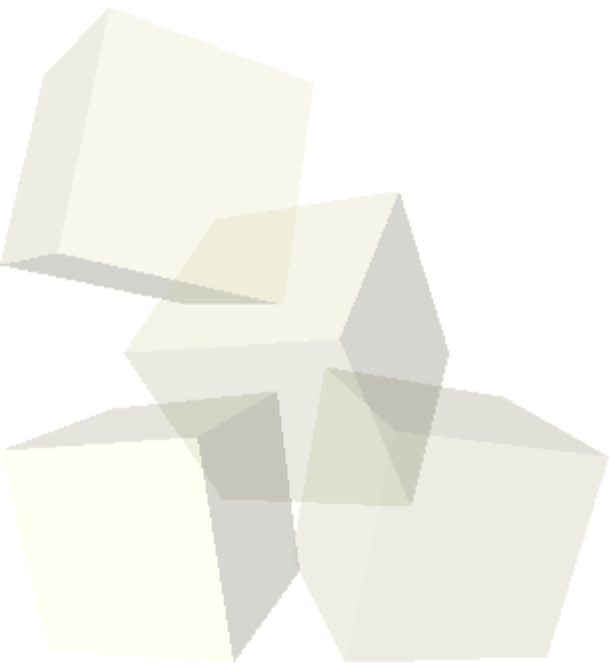


- Urvater aller Prozesse
- hat PID 1
- startet Programme nach /etc/inittab
 - ◆ Runscripts
 - ◆ getty – stellt I/O für virtuelle Terminals bereit, auf denen dann die Shell läuft
- kann direkte Child-Prozesse am Leben halten
- gut zu sehen mit „pstree“





- viele Skripte, jeweils für eine bestimmte Aufgabe zuständig
 - ◆ Partitionen mounten
 - ◆ Dateisysteme überprüfen (fsck)
 - ◆ Dienste (Daemons) starten
 - ◆ Netzwerkverbindungen Starten
 - ◆ eigene Startskripte



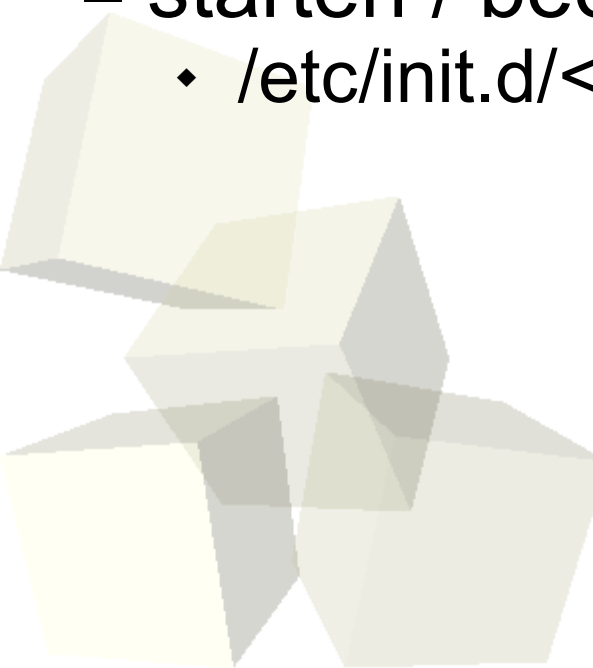


- verschiedene Systemzustände
- Wechsel mit *init runlevel*
- unterscheiden sich in Distributionen, z.B:
 - 1 Single User Mode
 - 2 kein Netzwerk
 - 3 normaler Runlevel mit Netzwerk
 - 5 mit GUI
 - 6 Reboot
 - 0 Halt
- 1, 6 und 0 sind in jeder Distribution gleich
- Debianderivate: normaler Betrieb in Runlevel 2



Runscripts / Startscripts

- Distributionen unterscheiden sich
- befinden sich in `/etc/init.d` oder `/etc/rc.d/init.d`
- welche gestartet werden, wird durch symbolische Links festgelegt
 - ♦ `/etc/rc*`
 - ♦ Debianderivate: `/etc/rcX.d/S<zahl><Dienst>`
- starten / beenden von Diensten in Debian
 - ♦ `/etc/init.d/<dienstname> start | stop | ...`

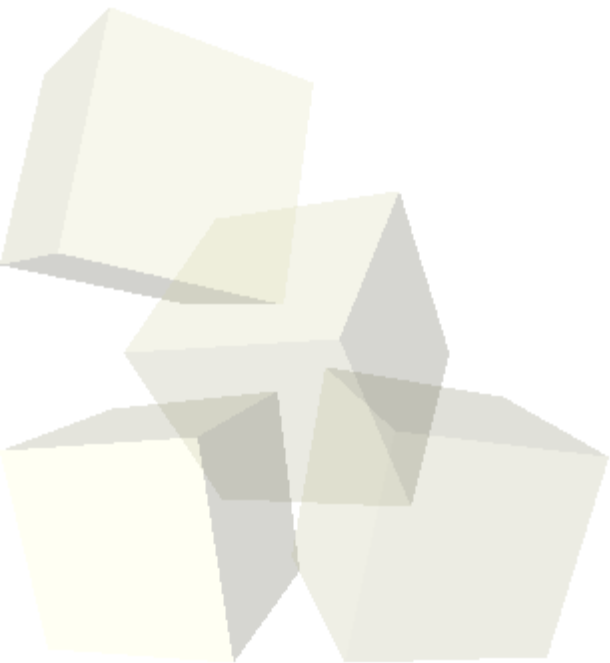


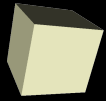


Ende des Bootvorgangs

- init startet getty auf den angegebenen virtuellen Terminals
- getty startet *login*

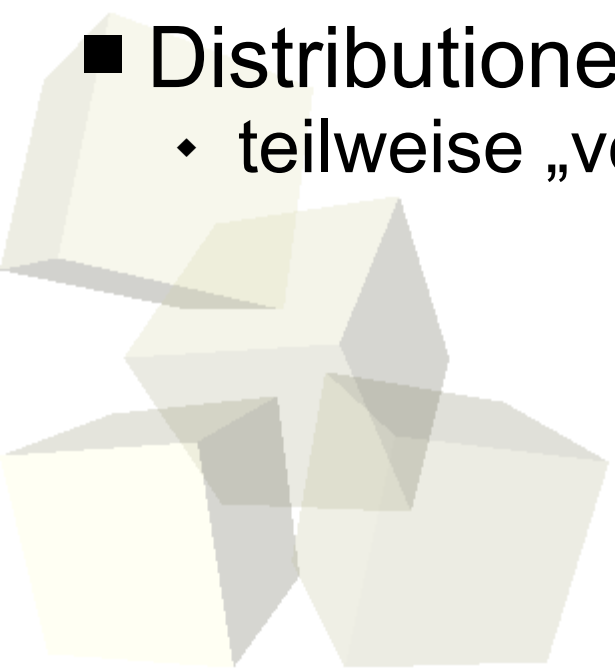
- Loginbildschirm wird angezeigt





Eigenen Kernel bauen

- Warum?
 - ◆ Sicherheitspatches hinzufügen
 - ◆ unnötige Teile entfernen (weniger Code, weniger Fehler)
 - ◆ Hardwaretreiber einbinden
 - ◆ Keine Module zur Erhöhung der Sicherheit
- Distributionsunabhängiger Kernel-Quellcode auf www.kernel.org (sog. „vanilla“ Kernel)
- Distributionen meist mit modifizierten Kernen
 - ◆ teilweise „verpatcht“ und unbrauchbar





Eigenen Kernel bauen

- Vorgehensweise Kernel 2.6.x
 - ◆ Quellcode besorgen
 - ◆ entpacken (in /usr/src/linux)
 - ◆ Konfigurieren des Kernels mit einem der Tools:
 - make config (stellt Fragen)
 - make menuconfig (graphisch in Konsole) benötigt ncurses
 - make xconfig (für graphische Oberfläche)
 - ◆ kompilieren
 - make
 - ◆ einbinden in Bootloader
- Konfiguration in .config gespeichert
- wird bei den ersten Versuchen evtl. nicht booten :-)



Eigenen Kernel Bauen

- neuer Kernel liegt in `/usr/src/linux/arch/i386/boot/bzImage`
 - ◆ `bzImage` ist der Kernel!
- wenn `.config` von altem Kernel vorhanden ist
 - ◆ Debian: in `/boot/config*` zu finden
 - ◆ angeblich auch in `/proc/config.gz`
 - ◆ `make oldconfig` (wendet alte Konfiguration auf neuen Kernel an = weniger Arbeit beim konfigurieren)
- Wichtige Hinweise
 - ◆ Dateisysteme immer in Kernel einbinden
 - ◆ bei „Kernel Panic“ in Internetsuchmaschinen und Foren nachforschen, meist gibt es eine einfache Lösung



Kernel Patchen

```
root@testkiste:/usr/src/linux# patch -p1 < ../netfilter-layer7-v2.16/kernel-  
2.6.22-2.6.23-layer7-2.16.patch
```

```
patching file include/linux/netfilter/xt_layer7.h
```

```
patching file include/net/netfilter/nf_conntrack.h
```

```
Hunk #1 succeeded at 128 (offset 1 line).
```

```
patching file net/ipv4/netfilter/nf_conntrack_l3proto_ipv4_compat.c
```

```
Hunk #1 succeeded at 163 (offset 5 lines).
```

```
patching file net/netfilter/Kconfig
```

```
Hunk #1 succeeded at 603 (offset -30 lines).
```

```
patching file net/netfilter/Makefile
```

```
Hunk #1 succeeded at 68 (offset -3 lines).
```

```
patching file net/netfilter/nf_conntrack_core.c
```

```
Hunk #1 succeeded at 330 with fuzz 1 (offset 123 lines).
```

```
patching file net/netfilter/nf_conntrack_standalone.c
```

```
Hunk #1 succeeded at 184 with fuzz 2 (offset 5 lines).
```

```
patching file net/netfilter/regexp/regexp.c
```

```
patching file net/netfilter/regexp/regexp.h
```

```
patching file net/netfilter/regexp/regmagic.h
```

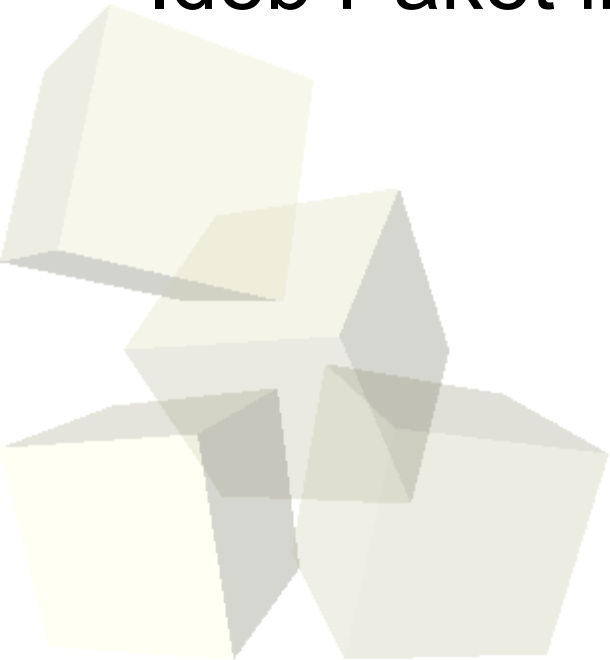
```
patching file net/netfilter/regexp/regsub.c
```

```
patching file net/netfilter/xt_layer7.c
```



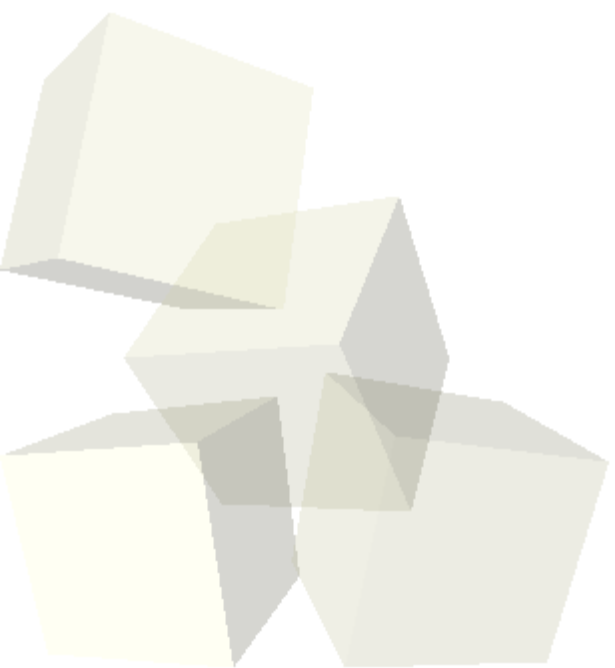
Kernel the Debian way

- Quellcode holen
 - ◆ aptitude install linux-source-2.6
 - ◆ liegt in /usr/src
- entpacken und nach /usr/src/linux verlinken
- konfigurieren
- kompilieren inkl. initrd Erstellung
 - ◆ make-kpkg kernel-image --initrd
- .deb Paket installieren





uebung_04_kernel.pdf





Das /proc Pseudodateisystem

- Informationen über laufende Prozesse
 - ♦ `/proc/PID/cmdline` - Kommandozeile
- Informationen über den Kernel
 - ♦ `/proc/version` - Versionsnummer
- Informationen über das System
 - ♦ `/proc/cpuinfo`
 - ♦ `/proc/meminfo`
- Einstellungen für den Kernel
 - ♦ `/proc/sys/net/ipv4/tcp_syncookies` - Syncookies
- Vergleich Anzahl Prozesse aus /proc mit „ps“
 - ♦ `ls -ld /proc/* | grep [0-9] | wc -l; ps ax | wc -l`



Befehls- und Ausgabeumleitung

- manchmal will man die Ausgabe eines Befehls in eine Datei oder wo anders hin umleiten
- manchmal will man die Eingabe eines Befehls aus einer Datei lesen
- manchmal will man die Ausgabe eines Befehls an einen Anderen füttern
- das geht ganz leicht... (mehr oder weniger sogar in Windows...)





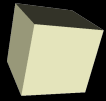
Ein paar Unix-Internals

■ 3 Standard-Filedescriptoren

- ◆ Standardeingabe: stdin
 - meist Tastatur
- ◆ Standardausgabe: stdout
 - meist Bildschirm
- ◆ Standardfehlerausgabe: stderr
 - meist ebenfalls Bildschirm
 - Trennung im Hinblick auf Umleitung sinnvoll

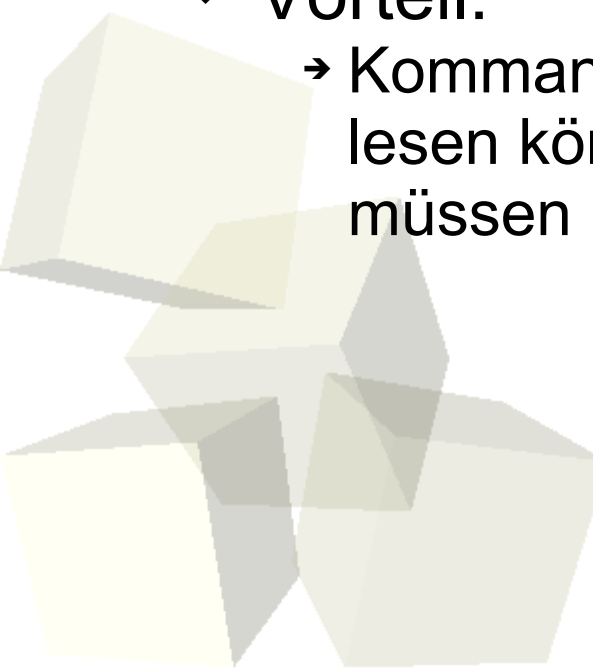
■ sind durchnummeriert

- ◆ STDIN 0
- ◆ STDOUT 1
- ◆ STDERR 2



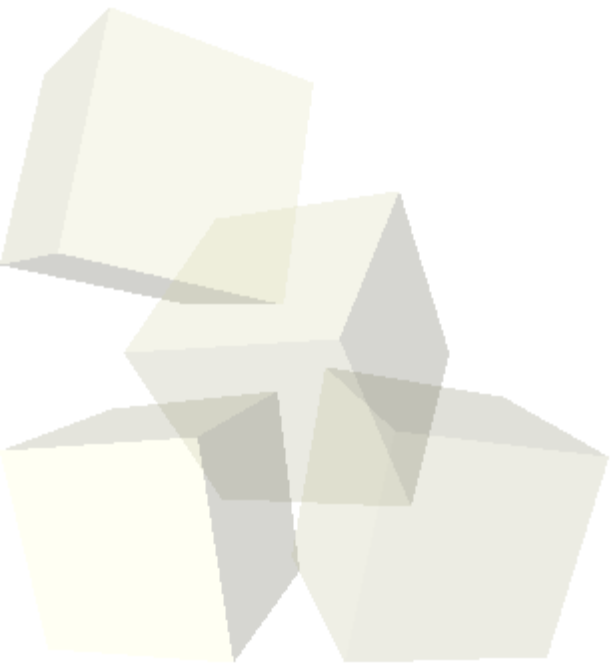
Standardfiledescriptoren

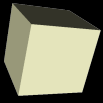
- sind automatisch offen / vorhanden
- viele Funktionen greifen automatisch auf diese zu
- Filedescriptoren können fast beliebig umgeleitet werden
 - ◆ in Dateien
 - ◆ an andere Programme
 - ◆ Vorteil:
 - Kommandos potentiell einfacher, da sie nur von *stdin* lesen können müssen und nur an *stdout* ausgeben müssen



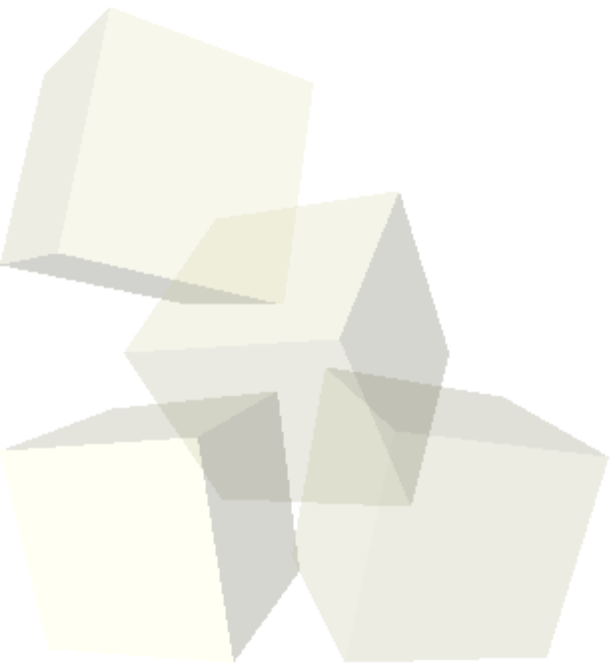


- Ergebnis wird in Datei geschrieben, nicht auf Terminal
- Umleitung erfolgt durch „>“ bzw. „>>“
 - ♦ „>“ fängt neue Datei an
 - ♦ „>>“ hängt an Bestehendes an
- *kommando > dateiname*
 - ♦ *cat /etc/fstab > /tmp/fstab.kopie*





- Programm liest aus Datei, nicht von der Tastatur
- Umleitung erfolgt durch „<“
- *kommando < dateiname*
 - ◆ *more < /etc/fstab*

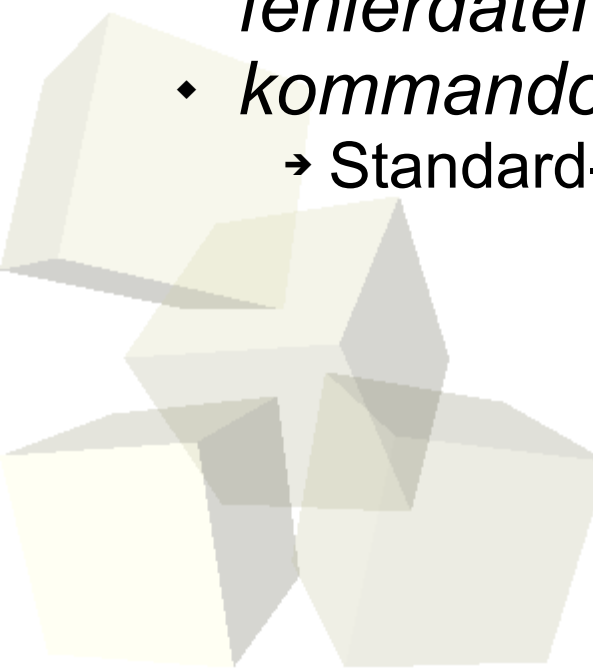




Umleitung der Fehlerausgabe

- Fehlermeldungen in Datei schreiben
- Umleitung erfolgt durch „2>“
- *kommando 2> dateiname*

- Kombinationen
 - ♦ *kommando < eingabedatei > ausgabedatei 2> fehlerdatei*
 - ♦ *kommando > datei 2>&1*
 - Standard- und Standardfehlerausgabe in datei





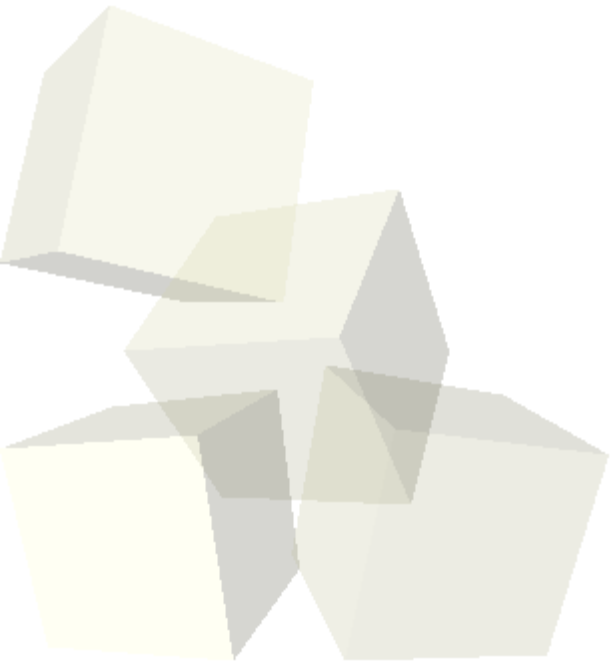
- Everything is a File
- Spezialziele:
 - ◆ /dev/null - schwarzes Loch für arme Bits
 - ◆ /dev/zero - lauter Null-Bytes (0x00)
 - ◆ /dev/random - Zufallsbytes
 - ◆ /dev/urandom - weniger zufällig
- diverse andere Gerätedateien
 - ◆ Umleitung direkt auf Geräte möglich
 - `cat /dev/zero > /dev/sdb1`





- verbinden Ausgabe eines Programms mit Eingabe eines anderen Programms über temporären Puffer
- mehrere Befehle können hintereinander geschaltet werden
- *befehl | befehl2 | befehl3*

- *ls -la /etc/ | more*





```
$ echo kitten > meow
```

```
$ echo "feline" >> meow
```

```
$ cat >> meow
```

```
tomcat
```

```
^D (bedeutet STRG+D)
```

```
$ cat < meow
```

```
kitten
```

```
feline
```

```
tomcat
```

```
$ grep cat < meow > cats
```

```
$ cat cats
```

```
tomcat
```

```
$ cat < meow |grep feline >> cats
```

```
$ grep kitten meow >> cats
```

```
$ < meow cat
```

```
kitten
```

```
feline
```

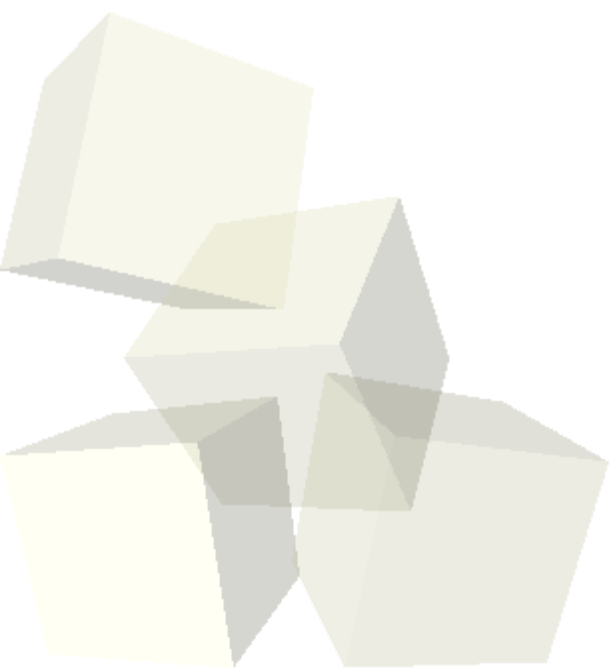
```
tomcat
```

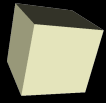
quelle nostromo:

http://nostromo.joeh.org/content/fhs/kwm06/cor3/cor3_linux_003_flowcontrol_and_redirects.pdf

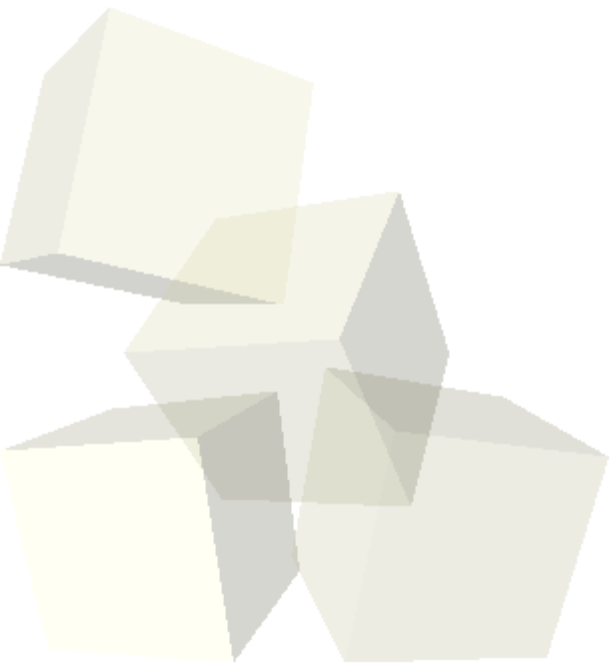


uebung_05_umleitung.pdf





- eine Shell kann mehrere Jobs gleichzeitig verwalten
- Jobs sind einzelne Prozesse
- nur ein Job kann im „Vordergrund“ laufen
 - ◆ Vordergrund = Zugriff auf das Terminal
- beliebig viele können im Hintergrund laufen





- jobs gibt Jobliste aus
 - Strg+z stoppt Ausführung und gibt Job-ID aus
 - fg %*id* hold Job mit *id* in den Vordergrund
 - bg *id* lässt Job im Hintergrund laufen
-
- *cmd1* *cmd1* im Vordergrund starten
 - *cmd1* & *cmd1* im Hintergrund
 - *cmd1* | *cmd2* Ausgabe von *cmd1* in *cmd2* pipen
 - *cmd1* ; *cmd2* nacheinander Ausführen
 - *cmd1* && *cmd2* *cmd2* nur wenn *cmd1* erfolgreich
 - *cmd1* || *cmd2* *cmd2* nur wenn *cmd1* nicht erfolg.



Beispiel in tcsh

```
$ view /etc/fstab &
```

```
[1] 5097
```

```
$ vi firewall.sh ^Z
```

```
[1] - Suspended (tty output)
```

```
view /etc/fstab
```

```
$ ping paranoid.org ^Z
```

```
Suspended
```

```
updatedb &
```

```
[4] 7725
```

```
$jobs
```

```
[1] Suspended (tty output)
```

```
view /etc/fstab
```

```
[2] - Suspended
```

```
vi firewall.sh
```

```
[3] + Suspended
```

```
ping paranoid.org
```

```
[4] Running
```

```
updatedb
```



- stellt virtuelle Terminals bereit
- laufen weiter, wenn SSH-Session beendet
- copy & paste ohne Maus
- Umgang mit screen
 - ◆ *screen* startet virtuelles Terminal
 - ◆ *screen -ls* zeigt screen IDs an
 - ◆ *screen -r <ID>* in screen-Session verbinden
 - ◆ *screen <befehl>* Befehl direkt in screen ausführen

 - ◆ *screen -S name* Session mit name erstellen
 - ◆ *screen -r name* auf Session name verbinden
- Befehle in screen
 - ◆ *strg+a+d* screen-Session verlassen (läuft)
 - ◆ *strg+d* screen-Session beenden



- jeder Shell hat eigene Skriptsprache
 - ◆ wir werden BASH-Skripting behandeln
- großer Funktionsumfang
- keine ausgewachsene Programmiersprache
- statt interaktiver Eingabe können Kommandos auch in einer Datei vorliegen

- Hello-World Skript

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

```
echo "Hello World!"
```



Vor- und Nachteile von Skripten

- müssen nicht kompiliert werden, direkt ausführbar
- sind langsamer als binäre Programme
- im Umgang mit Dateien und Systembefehlen sehr praktisch, da Shell dafür konzipiert
- leicht zu bearbeiten, keine IDE nötig
- teilweise gewöhnungsbedürftige Syntax
- Shell ist immer da, keine Installation zusätzlicher Interpreter (z.B. Perl) nötig



Erstellen von Shellskripten

- mit einfachem Texteditor
- beginnt mit Angabe der Shell
 - ◆ `#!/bin/bash`
- Execute-Recht auf Datei setzen
 - ◆ `chmod +x <Datei>`

■ Hello-World Skript

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

```
echo "Hello World!"
```

- `#`

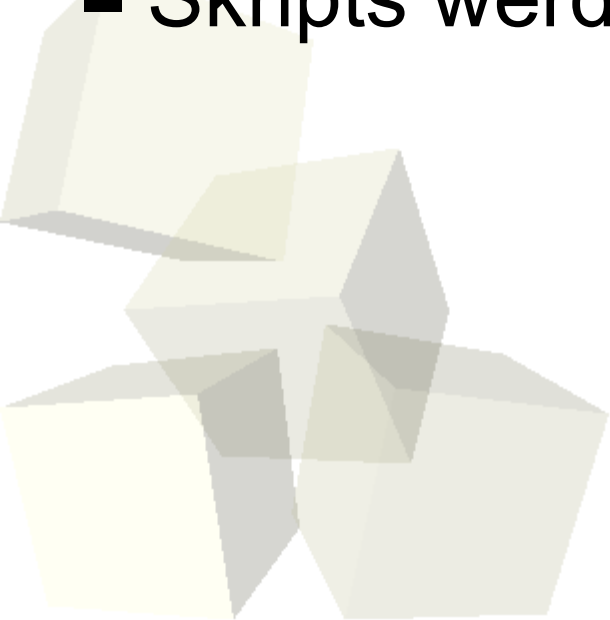
- ``Systembefehl``

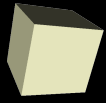
Leitet Kommentar ein
Backticks zum Ausführen
von Systembefehlen



Arten der Ausführung

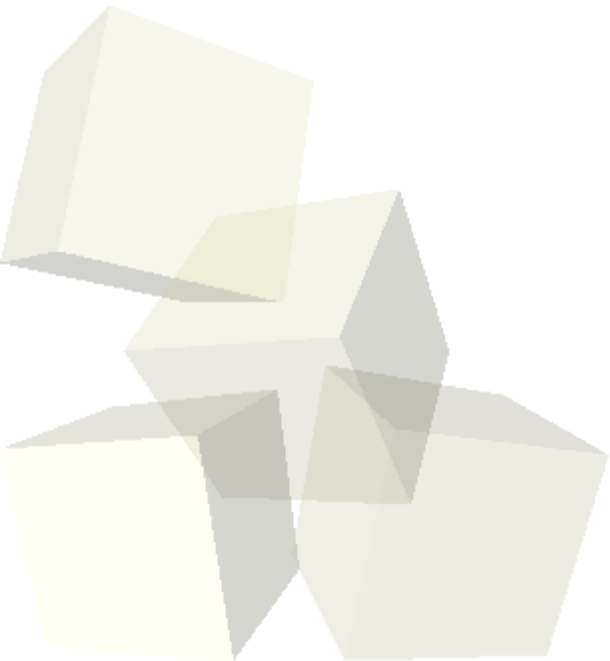
- mit Interpreter:
 - ◆ `bash /pfad/zum/skript`
 - ◆ Skript muss nicht ausführbar sein in diesem Fall
- wie normales Programm mit Pfad
 - ◆ `./skript`
 - ◆ Datei muss ausführbar sein
 - ◆ Interpreter wird von Kernel gesucht anhand der ersten Zeile
- Skripts werden immer in eigener Shell ausgeführt





Spezielle Variablen

- Stehen in jedem Shellskript zur Verfügung
- \$0 - Name des Skripts
- \$1-9 - 1-9tes Argument
- \$# - Anzahl der Argumente
- \$@ - alle Argumente



Beispiele für Argumentauswertung

■ beispiel.sh

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

```
echo "Mein Name ist $0"
```

```
echo "Es wurden $# Argumente übergeben"
```

```
echo "1. Argument: $1"
```

```
echo "2. Argument: $2"
```

```
echo "3. Argument: $3"
```

```
echo "alle zusammen: $@"
```

■ Aufruf mit

- ♦ ./beispiel.sh arg1 arg2 arg3



Bedingungen testen

- `test bedingung oder [bedingung]`
 - ♦ prüft *bedingung* und liefert 0 wenn wahr, 1 wenn falsch und 2 falls *bedingung* einen Fehler enthält

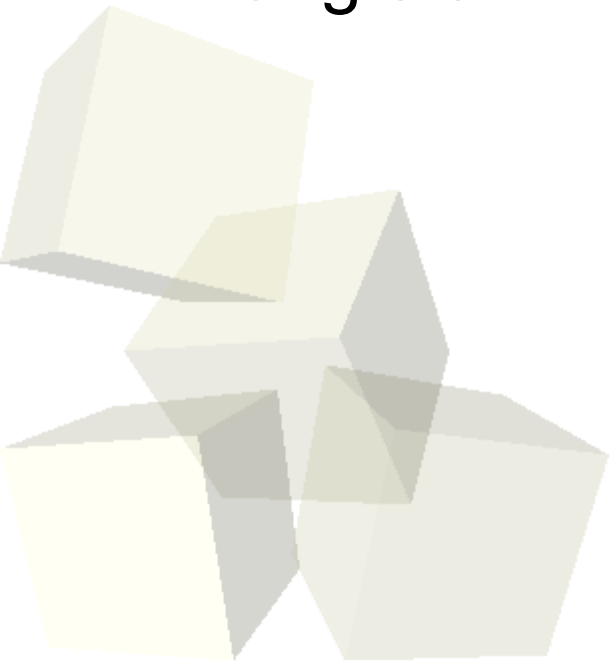
- **Beispiel**
 - ♦ `test -w /etc/passwd`
 - ♦ `[-w /etc/passwd]`
 - ♦ prüft ob `/etc/passwd` beschreibbar ist

- `test -f <datei>`
 - ♦ prüft ob existiert und reguläre Datei
- `test -d <verzeichnis>`
 - ♦ prüft ob existiert und Verzeichnis



Vergleiche und Verknüpfungen

- `test ausdruck1 -a ausdruck2`
 - ♦ logisches UND
- `test ausdruck1 -o ausdruck2`
 - ♦ logisches ODER
- `test ausdruck1 -eq|ge|gt|le|lt ausdruck2`
 - ♦ gleich, größer gleich, größer, kleiner gleich, kleiner
- `test ausdruck1 -ne ausdruck2`
 - ♦ ungleich





- users.sh

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

```
USERS=`who | wc -l`
```

```
if [ $USERS -lt 5 ]
```

```
then
```

```
    echo "Weniger als 5 Benutzer angemeldet"
```

```
else
```

```
    echo "Mehr als 5 Benutzer angemeldet"
```

```
fi
```

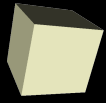
- Achtung: zwischen [und] Klammern ein Leerzeichen setzen!

- alternativ:

```
if test $USERS -lt 5
```

```
then
```

```
....
```



■ for1.sh

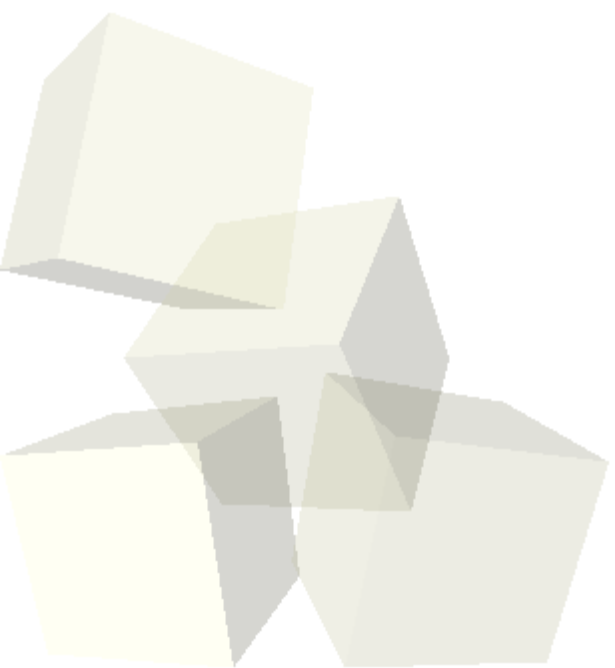
```
#!/bin/bash
for x in hans heinz karl luise
do
    echo " Name: $x"
done
```

■ for2.sh

```
#!/bin/bash
for datei in *
do
    cat $datei
done
```



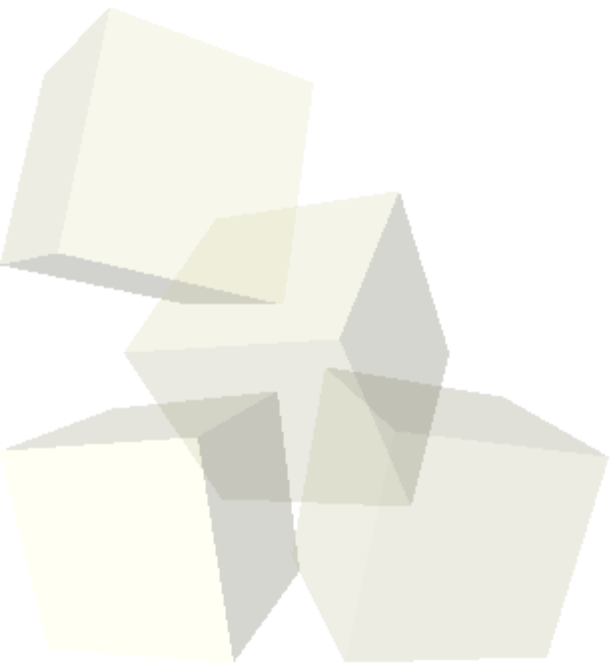
```
■ while.sh
#!/bin/bash
while [ 1 -eq 1 ]
do
    echo „little DoS attack“
done
```





Berechnungen durchführen

- mit Hilfe von *expr*
- Punktrechnung vor Strichrechnung
- Beispiel
 - $x = \text{`expr \$A + \$B + \(2 * 3 \)`}$



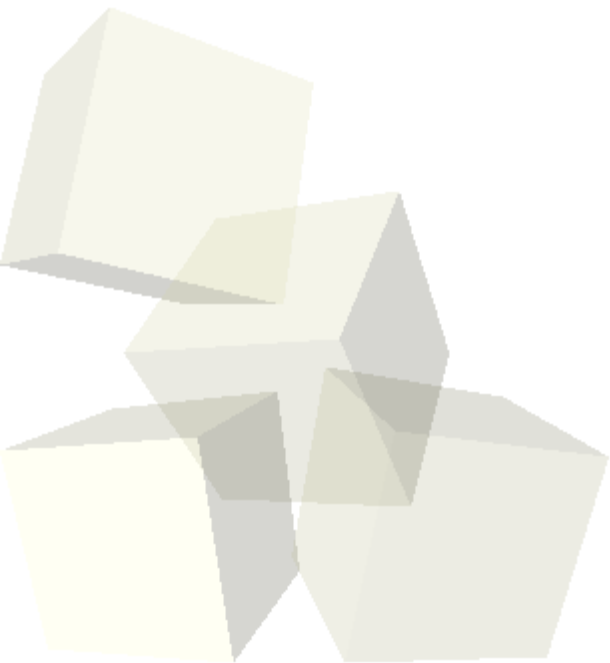


■ Wissensquellen

- ◆ <http://www.linuxfibel.de/bashprog.htm>
- ◆ http://www-user.tu-chemnitz.de/~hot/unix_linux_werkzeugkasten/
- ◆ <http://www.tldp.org/LDP/abs/html/>
- ◆ <http://www.tldp.org/HOWTO/Bash-Prog-Intro-HOWTO.html>

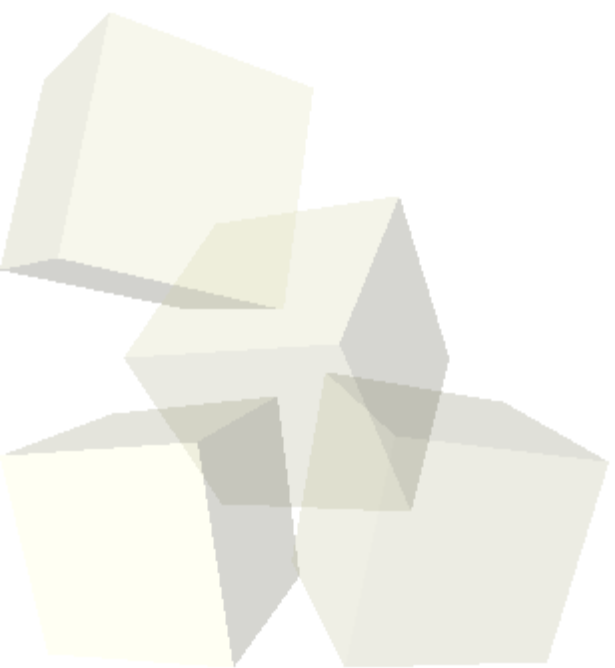
■ Skriptsammlungen

- ◆ <http://www.shelldorado.de>





uebung_06_shell-programmierung.pdf





■ ethtool

- ◆ zeigt Statusinformationen über Netzwerkkarten

■ tcpdump

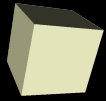
- ◆ Netzwerksniffing-Tool
- ◆ `tcpdump -i eth1 -n port 80`
- ◆ `tcpdump icmp`

■ ntpclient / ntpd

- ◆ Zeitsynchronisation
- ◆ `ntpdate ntp1.fau.de ptbtime1.ptb.de de.pool.ntp.org`
- ◆ ntpd sorgt für kontinuierlich (richtige) zeit - VMware!

■ dig

- ◆ DNS Tool
- ◆ `dig @nameserver domain a`
 - A-Records Domain bei „nameserver“ erfragen
- ◆ `dig -x 85.10.199.44`
 - reverse lookup



- **autossh**
 - ◆ überwacht SSH Tunnel
 - ◆ verwendet SSH Syntax
- **ddclient**
 - ◆ dyndns Client mit vielen Features
- **hdparm**
 - ◆ Geschwindigkeit von HDDs testen und Einstellen
 - ◆ `hdparm -tT /dev/sda`
- **hdtemp**
 - ◆ Temperaturanzeige HDD
- **logwatch**
 - ◆ tägliche Logauswertung, mieser Konfiguration
- **memtest86**
 - ◆ das Tool schlecht hin zum Speicher testen



- im Linuxkernel integrierter Paketfilter
- sehr mächtig
- standardmäßig bis Layer 4
- Projekt mit Layer 7 Filterung
- durch Module erweiterbar
 - ◆ Log, Mark usw.
- Steuerung über *iptables* Befehl
- Firewallregeln als Shellskript schreiben
 - ◆ Variablen verwenden für Interfaces, Pfade usw.
 - ◆ leichte Portierbarkeit auf andere Systeme

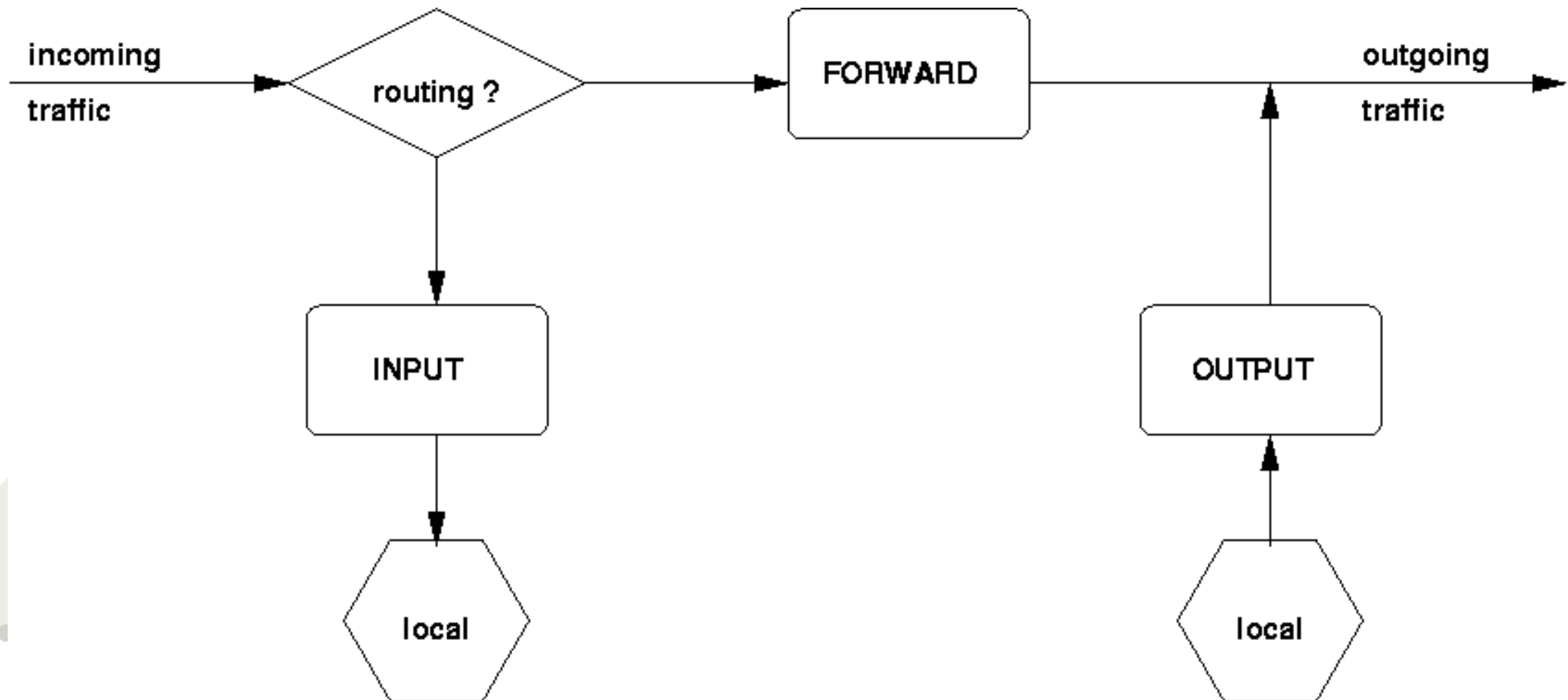


- alle Befehle direkt eingeben
- grafische Tools zur Erstellung
 - ◆ Firestarter
 - ◆ Firewall-Builder
 - ◆ ...
- internetbasierte Iptables-Generatoren
 - ◆ <http://easyfwgen.morizot.net/gen/>
 - ◆ ...
- Empfehlung: einmal damit beschäftigen und von Hand erstellen





Übersicht

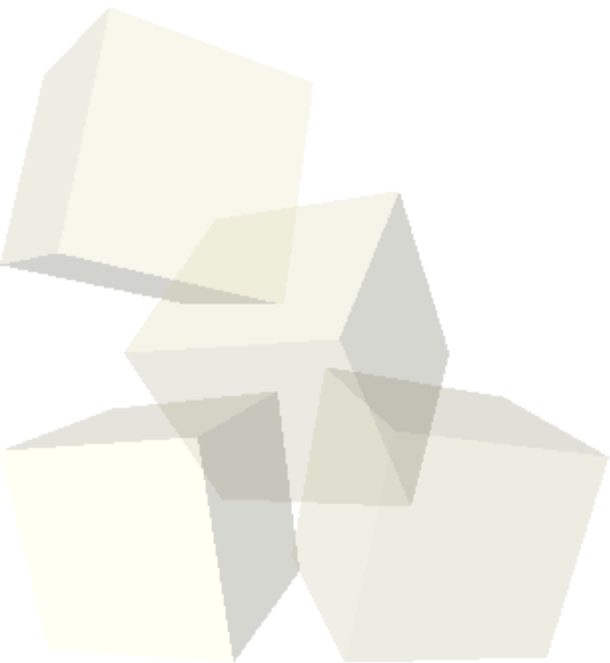


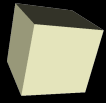
Quelle: <http://www.tu-chemnitz.de/urz/kurse/unterlagen/iptables.html>



Aufbau Iptables Skript

- Variablen definieren (Netzwerkkarten, Pfade)
- nötige Module laden lassen
- Kerneinstellungen in /proc
- alte Regeln löschen, Tabellen flushen
- Default Policy setzen
- Connection Tracking aktivieren
- eigentliche Firewallregeln





```
iptables -F
```

```
iptables -t nat -F
```

```
iptables -t mangle -F
```

```
iptables -X
```

```
iptables -t nat -X
```

```
iptables -t mangle -X
```

```
# default policy: drop everything
```

```
iptables -P INPUT DROP
```

```
iptables -P FORWARD DROP
```

```
iptables -P OUTPUT DROP
```

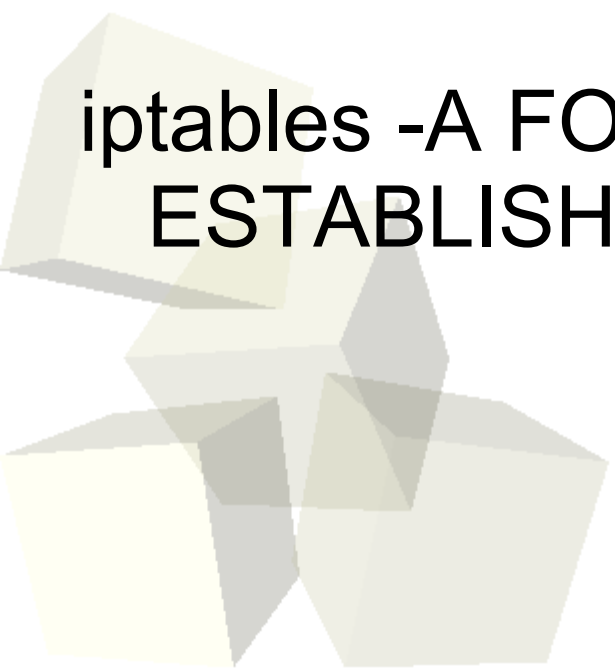


- auch Connection Tracking genannt

```
iptables -A INPUT -m state --state \
    ESTABLISHED,RELATED -j ACCEPT
```

```
iptables -A OUTPUT -m state --state \
    ESTABLISHED,RELATED -j ACCEPT
```

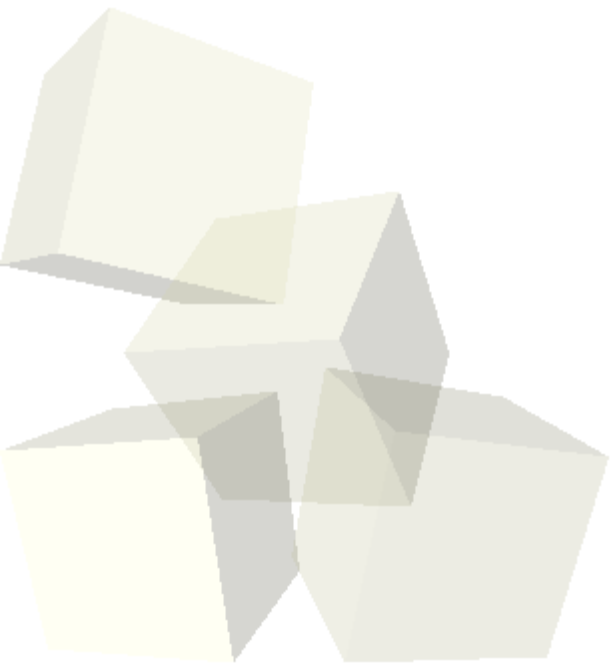
```
iptables -A FORWARD -m state --state \
    ESTABLISHED,RELATED -j ACCEPT
```



Loopback Interface alles zulassen

```
iptables -A INPUT -i lo -j ACCEPT
```

```
iptables -A OUTPUT -o lo -j ACCEPT
```



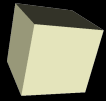


- `iptables -A INPUT -d $IP -p tcp --dport 22 -j ACCEPT`
 - ◆ Regel am Ende der INPUT Kette einfügen
 - ◆ ssh akzeptieren, wenn \$IP angesprochen wird

- `iptables -I OUTPUT -p udp --dport 53 -j DROP`
 - ◆ Regel am Anfang der OUTPUT Kette einfügen
 - ◆ DNS blockieren

- `iptables -t nat -A POSTROUTING -o $EXT -j MASQUERADE`
 - ◆ Masquerading aktivieren

- `iptables -A FORWARD -i $INT -o $EXT -p tcp --dport 80 -j ACCEPT`
 - ◆ HTTP von intern nach extern weiterleiten



- `iptables -t nat -A PREROUTING -p tcp -i $EXT -d 192.168.1.20 --dport 3389 -j DNAT --to 192.168.0.2:3389`

```
iptables -A FORWARD -i $EXT -o $INT -d 192.168.0.2 -p tcp --dport 3389 -j ACCEPT
```

- ♦ bei NAT von Extern nach Intern zugreifen – 2 Regeln nötig!

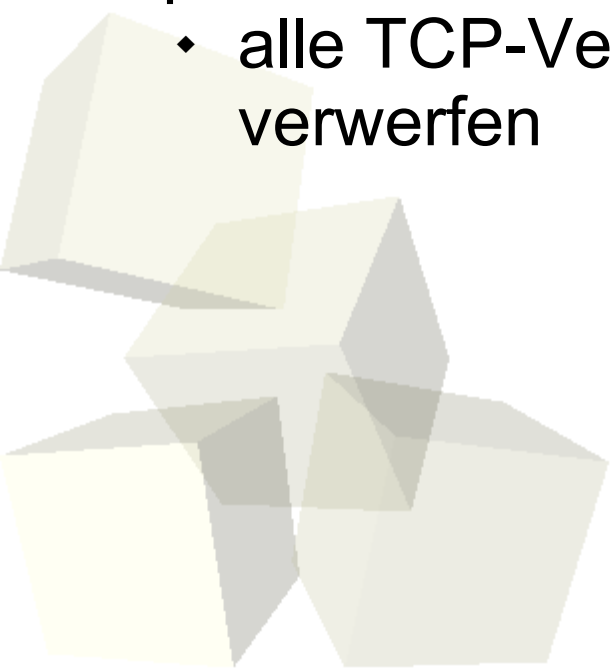
- `iptables -t nat -A PREROUTING -i $EXT -d $IPADDRESS -p tcp --dport 5521 -j DNAT --to $IPADDRESS:21`

- ♦ alle Anfragen an Port 5521 an Port 21 übergeben
- ♦ keine Übergabe an fremde IP-Adressen möglich (Rückweg würde fehlen)



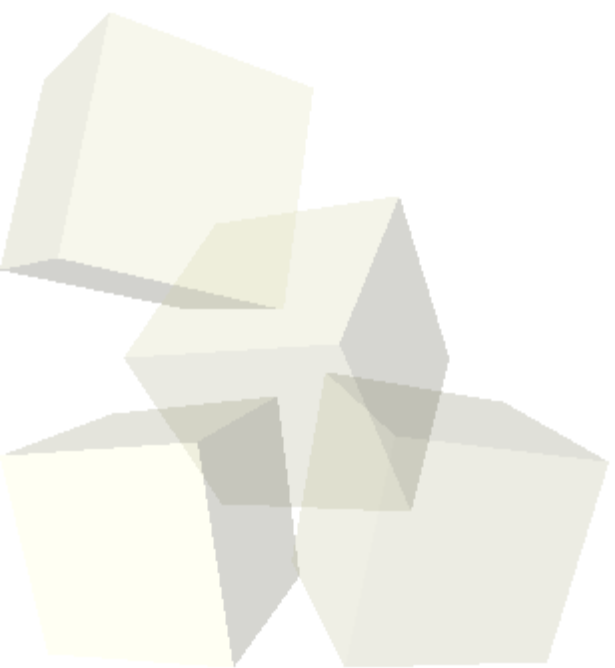
Beispiele Spezialaufgaben

- `iptables -A INPUT -s www.slashdot.org -j DROP`
 - ♦ IP wird nur einmal bei Ausführung aufgelöst, nicht bei jedem Paket!
- `iptables -I INPUT -p icmp -m limit ---limit 4/s --limit-burst 5 -j ACCEPT`
 - ♦ vier bis fünf ICMP Pakete pro Sekunde zulassen
- `iptables -A INPUT -p tcp ! --syn -m state --state NEW -j DROP`
 - ♦ alle TCP-Verbindungen, die nicht mit SYN beginnen verwerfen





uebung_07_iptables.pdf



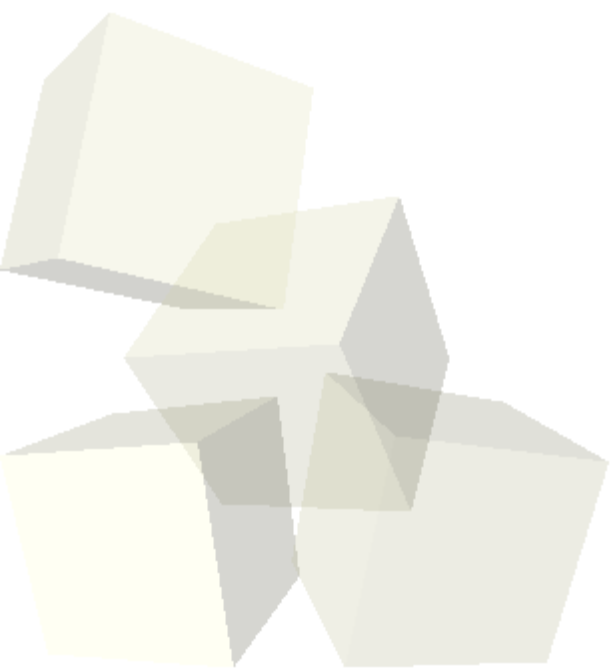


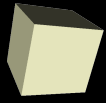
Vorgehensweise zur Installation eines Dienstes - Webserver

- Welche Webserver gibt es?
- Welche unterstützen die benötigten Features?
- Welche Server sind in Distribution verfügbar?
- Wo finde ich Anleitungen / HowTos?
- **vor** der Installation Anleitungen lesen
- Dienst durch Firewall vorläufig blockieren
- Installation
- Testen ob alles gewünschte geht
- in kleinen Schritten vorgehen



uebung_08_webserver.pdf





- [1]
http://www.ssh.com/support/documentation/online/ssh/adminguide/32/Port_Forwarding.html
- Die Folien sind an <http://www.topfen.net/linux/> angelehnt

