Technische Praxis der Computersysteme

Funktionen:

Funktionen werden via `name () {...}` oder `function name {...}` definiert. Optional kann bei der Form `function name` auch eine `()` genutzt werden.
Aus Funktionen kann mit `return [n]` wieder zurückgekehrt werden. Der Statuswert ist dabei optional.

Auflisten und Löschen von Funktionen:

Eine Liste aller in einer Shell definierten Funktionen samt Code erhält man mit `typeset -f`.
Eine einzelne Funktion kann dabei mit `typeset -f name` betrachtet werden.
Möchte man eine definierte Funktion wieder löchen kann dies mit `unset -f name` gemacht werden.

Funktionsargumente:

Funktionsargumente werden - anders als bei vielen Programmiersprachen - nicht explizit in der Definition festgelegt, sondern "einfach" übergeben, i.e. `name arg1 arg2... argn` ruft `name` mit *n* Argumenten `arg1 arg2... argn` auf, auf die dann wie bei Shellscripts via `$1,..,$n` zugegriffen werden kann. Auch die speziellen Variablen `$@`, `$*`, `$#` und `$0` werden entsprechend umgesetzt.

Function body enclosure:

Es gibt zwei verschiedene Arten den Code einer Funktion zu umschließen. Üblicherweise nutzt man dafür `{...}`, man kann aber auch `(...)` verwenden. Der Code zwischen `{...}` wird von der laufenden Shell interpretiert, während der Code zwischen `(...)` in einer Subshell ausgeführt wird.

Function body enclosure:

Es gibt zwei verschiedene Arten den Code einer Funktion zu umschließen. Üblicherweise nutzt man dafür `{...}`, man kann aber auch `(...)` verwenden. Der Code zwischen `{...}` wird von der laufenden Shell interpretiert, während der Code zwischen `(...)` in einer Subshell ausgeführt wird.

Bemerkung:

- Insbesondere kann der Code in `(...)` die Variablen der Parent Shell nicht nachhaltig modifizieren.
- `{...}` sowieso `(...)` werden ganz allgemein zum Gruppieren von Befehlen genutzt und können an beliebiger Stelle eingesetzt werden.

alias:

"Aliases allow a string to be substituted for a word when it is used as the first word of a simple command."
Ein Alias kann mit `alias name="complex command"` definiert und mit `unalias` wieder entfernt werden. Setzt man z.B. `alias ls="ls -alh"`, so wird jeder `ls` Aufruf in der Shell durch `ls -alh` ersetzt.

alias:

"Aliases allow a string to be substituted for a word when it is used as the first word of a simple command."
Ein Alias kann mit `alias name="complex command"` definiert und mit `unalias` wieder entfernt werden. Setzt man z.B. `alias ls="ls -alh"`, so wird jeder `ls` Aufruf in der Shell durch `ls -alh` ersetzt.

Bemerkung:

Manche Shells - z.B. die `bash` - ignorieren `alias` in Skripten.

Einbinden von Dateien in eine Shell:

Eine Shell bzw. ein Shellscript kann auch andere Dateien einbinden und so z.B. Funktionen und Variablen aus anderen Dateien importieren. Dazu wird die zu importierende Datei als Argument an den `source` Befehl übergeben, e.g.

```
source ~/.zsh/functions
```

Standarddateien, die von Shells einbunden werden:

Die meisten Shells importieren zu Beginn der Ausführung einige Standarddateien z.B. `~/.bashrc` oder `~/.zshrc` für *interaktive* Varianten der `bash` bzw. `zsh`. Dort kann man Befehle ablegen, die man bei jedem Startup ausführen möchte. Typischerweise betrifft das vor allem das Setzen von Variablen, Aliases, Funktionen, Einstellungen,...

Standarddateien, die von Shells einbunden werden:

Die meisten Shells importieren zu Beginn der Ausführung einige Standarddateien z.B. `~/.bashrc` oder `~/.zshrc` für *interaktive* Varianten der `bash` bzw. `zsh`. Dort kann man Befehle ablegen, die man bei jedem Startup ausführen möchte. Typischerweise betrifft das vor allem das Setzen von Variablen, Aliases, Funktionen, Einstellungen,...

Bemerkung:

Je nach Art der Shell und des Aufrufs (interaktiv, nicht interaktiv, loginshell) liest eine Shell oft andere bzw. spezielle Datein an. Login-Shells lesen typischerweise auch "`profile`" Dateien wie `/etc/profile`, `~/.profile` (bash) oder `/etc/zprofile`, `~/.zprofile` (zsh) ein. Interaktive Shells lesen üblicherweise die "`rc`"-files wie `/etc/bashrc`, `~/.bashrc` usw. ein. Die `zsh` liest *immer* `/etc/zshenv` ein.

Texteditoren:

Um Texte in einer Terminalumgebung zu editieren stehen - neben den üblichen Shellcommands, die wir schon gesehen haben - auch Texteditoren wie `vi`, `vim`, `emacs` oder andere zur Verfügung. Auf allen POSIX-Systemen ist üblicherweise ein `vi`-kompatibler Editor installiert.

Texteditoren:

Um Texte in einer Terminalumgebung zu editieren stehen - neben den üblichen Shellcommands, die wir schon gesehen haben - auch Texteditoren wie `vi`, `vim`, `emacs` oder andere zur Verfügung. Auf allen POSIX-Systemen ist üblicherweise ein `vi`-kompatibler Editor installiert.

Bemerkung

Die Frage nach einem guten Texteditor ist üblicherweise unter Informatikern in schwer religiösen Bereichen angesiedelt und damit die persönliche Wahl jedes Nutzers (siehe `http://xkcd.com/378/`). Wir beschränken uns hier schlichtweg aus Standardkonformität auf `vi(m)` (der Autor dieser Folien ist zufälligerweise überzeugter Vertreter der Nutzerschaft dieses Editors).

Grundlegends zu vi:

`vi` ist ein interaktiver full-screen Editor, der zwei grundlegende Modi kennt:

- Command Mode: Erlaubt die Eingabe von Befehlen
- Insert Mode: Editieren (eigentlich nur einfügen!) von Text.

`vi` kann einfach via `vi` oder `vi filename` aufgerufen werden.

Grundlegends zu vi:

`vi` ist ein interaktiver full-screen Editor, der zwei grundlegende Modi kennt:

- Command Mode: Erlaubt die Eingabe von Befehlen
- Insert Mode: Editieren (eigentlich nur einfügen!) von Text.

`vi` kann einfach via `vi` oder `vi filename` aufgerufen werden.

Bemerkung:

`vim` erlaubt wesentlich mehr Modi (insbesondere z.B. einen visual Mode um Text zu markieren...).

Command Mode:

Der Command Mode erlaubt eine Vielzahl von Befehlen, die meist auf einer geoeffneten Datei operieren. Grundlegende Befehle sind:

- `h, i, j, k`: Bewegt den Pointer ein Zeichen nach links, unten, oben, rechts.
- `i, a`: Wechselt in den Insert Mode beim/nach dem aktuellen Zeichen.
- `o, O`: Legt eine neue Zeile unter/über der aktuellen Zeile an und welchselt in den Insert Modus am Anfang der neuen Zeile.
- `u`: Undo

Command Mode:

- `:e filename <Return>`: Öffnet die Dateil `filename` zum Editieren.
- `:w <Return>`: Speichern ("write"). Um in eine (andere) Datei zu speichern kann optional ein Dateiname angegeben werden.
- `:q <Return>` Beenden. Mit `:q!` kann beendet werden ohne die Änderungen zu speichern.

Edit Mode:

Der Edit Mode dient der Eingabe von neuem Text. Um den Edit Mode wieder zu Verlassen dient `Esc`-Key.

Edit Mode:

Der Edit Mode dient der Eingabe von neuem Text. Um den Edit Mode wieder zu Verlassen dient `Esc`-Key.

Bemerkung:

Im Edit Mode kann man üblicherweise nicht navigieren. Einige `vi`-implementierungen (wie `vim`) erlauben es dennoch mit Hilfe der Pfeiltasten zu navigieren.

Navigationsbefehle:

- `0`: Zeilenanfang
- `^`: Erstes nicht-Leerzeichen der Zeile
- `$`: Zeilenende
- `w`,`W`: Anfang des nächsten Worts (`W` ignoriert Satzzeichen)
- `e`,`E`: Ende des nächsten Worts (`E` ignoriert Satzzeichen)
- `b`,`B`: Anfang des vorhergehenden Worts (`B` ignoriert Satzzeichen)
- `H`,`M`,`L`: Bewegt Pointer zur ersten, mittleren bzw. letzten Zeile *am Bildschirm*

Navigation II:

- `G`: Letzte Zeile in der Datei
- `nG`: n-te Zeile in einer Datei

L”oschen & Kopieren:

- `x`: Löscht Zeichen unter Pointer
- `dd`: Löscht aktuelle Zeile
- `dw`: Löscht das nächste Wort
- `yy`: Kopiert aktuelle Zeile
- `yw`: Kopiert nächstes Wort
- `p`: Einfügen nach Pointer

Bemerkung:

Die meisten Befehle (wie z.B. `x`, `dd`, `yy`, `hijk` oder `w`...) kann eine Zahl vorangestellt werden. In diesem Fall wird der Befehl auf die nächsten *n* "Einheiten" ausgedehnt, z.B. löscht `5x` die nächsten 5 Zeichen, `3yy` kopiert die nächsten 3 Zeilen usw.

vi und file handling:

Den oben vorgestellten Befehlen `:w`, `:q` und `:e` kann ein `!` nachgestellt werden, um Fehlermeldungen zu ignorieren. So beendet `:q!` vi ohne Änderungen in eine Datei zu schreiben. Außerdem gibt es noch

- `:x`: Beendet vi, schreibt nur falls Änderungen vorgenommen wurden.
- `:r file`: Fügt den Inhalt von `file` nach dem Cursor ein.
- `:r !command`: Fügt die Ausgabe von `command` nach dem Cursor ein.

Suchen von Text:

In `vi` kann mittels `/text` bzw. `?text` in einer Datei nach Text gesucht werden. `/` sucht dabei vorwärts, `?` rückwärts, also nach Suchergebnissen nach bzw. vor der aktuellen Position. `vi` unterstützt dabei Regular Expressions in `text`. Mit `n` und `N` kann man das nächste bzw. vorhergehende Suchergebnis anzeigen.

Suchen und Ersetzen von Text:

Search-and-Replace funktioniert in `vi` ähnlich zu `sed`. Selbst der zugehörige Befehl sieht analog aus:
`:s/pattern/replacement/flags`, wobei `pattern` eine RE sein kann und `flags` z.B. `g` sein können, um alle Vorkommnisse von `pattern` durch `replacement` zu ersetzen. Der `:s`-Befehl operiert normalerweise nur auf der *aktuellen Zeile*, kann aber, wie die meisten `:`-Befehle, mit einer *Range* versehen werden, z.B.:

- `n,m` Alle Zeilen zwischen `n` und `m`.
- `.` Aktuelle Zeile.
- `$` Letzte Zeile.
- `%` Alle Zeilen.
- `g/pattern/` Alle Zeilen, die `pattern` erfüllen.

Bemerkung:

Möchte man also alle Vorkommnisse einer Pattern in einer Datei Ersetzen muss man `%s/pattern/repl/g` nutzen (alle Vorkommnisse in allen Zeilen). Bei `n,m` können n bzw. m auch `.` oder $ sein. So kann man z.B. mit `.,$d` alle Zeilen ab der aktuellen Zeile bis zum Ende der Datei löschen.

vi und die passwd-Datei:

Möchte man die `/etc/passwd` Datei editieren sollte man den speziellen `vipw` Befehl nutzen. Auf manchen Systemen gibt es auch `vigr` für die `/etc/group` Datei, also um Gruppen anzupassen. `vipw` nutzt per default `vi` zum Editieren der `passwd`, dies kann aber über die `$EDITOR` Variable in der Shell geändert werden.

Systemdienste:

Auf einem System läeuft üblicherweise mehr als nur die Prozesse eines Nutzers. Auch finden sich in der `/etc/passwd` mehr als nur `root` und Useraccounts. Viele Nutzer sind reine *Systemnutzer*, also Nutzer, die vom System zum Bereitstellen div. "globaler" Dienste - der Systemservices - genutzt werden.

Systemdienste:

Auf einem System läeuft üblicherweise mehr als nur die Prozesse eines Nutzers. Auch finden sich in der `/etc/passwd` mehr als nur `root` und Useraccounts. Viele Nutzer sind reine *Systemnutzer*, also Nutzer, die vom System zum Bereitstellen div. "globaler" Dienste - der Systemservices - genutzt werden.

Ausblick:

Wir werden uns im folgenden vorerst mit zwei essentiellen Systemdiensten begnügen: `cron` und `ssh`. Beides sind essentielle Dienste und auf allen gängigen Systemen meist per default installiert.

crond:

`cron` ist eine Familie an Werkzeugen um Programme nach einen vorgegebenen - üblicherweise Periodischen - Muster automatisiert ausführen zu lassen (e.g. backups, checks,...). Essentieller Bestandteil ist der `crond`, ein *Daemon*, der `crontabs` einliest und darauf basierend Programme zu festgelegten Zeiten ausführt.

crontabs:

Das Format von `crontabs` ist in `crontab(5)` dokumentiert. Die `crontab` Datei kann mit dem Befehl `crontab -e` editiert werden. Jede Zeile ist dabei wie folgt aufgebaut:

```
*    *    *    *    *    command
|    |    |    |    |
|    |    |    |    +-   Tag der Woche (0-6),
|    |    |    |         0 = Sonntag
|    |    |    +-----   Monat (1-12)
|    |    +----------   Tag des Monats (1-31)
|    +---------------   Stunde (0-23)
+--------------------   Minute (0-59)
```

crontabs:

Die Zeitangaben in den ersten 5 Feldern können

- einzelne Zahlen,
- Schrittweite in der Form `n-m` sowie `n-m/l`, wobei `l` die Schrittweite angibt.
- `*` für jeden Wert, bzw. `*/l` für jeden `l`-ten Wert.
- Mit `,` getrennte Listen der obigen Einträge sein

crontabs:

Die Zeitangaben in den ersten 5 Feldern können

- einzelne Zahlen,
- Schrittweite in der Form `n-m` sowie `n-m/l`, wobei `l` die Schrittweite angibt.
- `*` für jeden Wert, bzw. `*/l` für jeden `l`-ten Wert.
- Mit `,` getrennte Listen der obigen Einträge sein

Bemerkung:

Für Tage und Monate kann man auch "Kurzbezeichnungen" in Form der ersten drei Buchstaben des jeweiligen Tages/Monats nutzen, e.g. `mon` für Montag, `tue` für Dienstag, usw.

Berechtigungen für cron:

Üblicherweise kann jeder Nutzer cronjobs anlegen. Es ist aber möglich dies über die Dateien `cron.allow` und `cron.deny` einzuschränken, die üblicherweise in `/etc/` liegen. Existiert `cron.allow` dürfen nur Nutzer, deren Name in dieser Datei steht, crontabs anlegen. Analoges gilt für `cron.deny`.

System cronjobs:

Die `crontab` des `root` Nutzers wird meistens als “system crontab” bezeichnet. Dort finden sich üblicherweise Einträge für alle Aufgaben, die von einem System regelmäßig durchgeführt werden müssen. Weit verbreitet ist es außerdem, im `/etc`-Verzeichnis die Verzeichnisse `cron.hourly`, `cron.daily`, `cron.weekly` und `cron.monthly` zum Ablegen von Shellscripts zu nutzen, die dann stündlich, täglich,... usw. ausgeführt werden.

cron und root:

User können nur ihre eigene `crontab` editieren. `root` Nutzer dürfen aber die `crontabs` aller Nutzer verändern. Dazu muss der Nutzername per `-u username` Parameter an den `crontab` Befehl übergeben werden.

openssh:

`ssh` bezeichnet eine Familie an Werkzeugen, um aus "der Ferne" (i.e. über ein Netzwerk) auf Rechner zugreifen zu können. `ssh` bezieht sich dabei meistens auf `openssh`, eine weit verbreitete, freie ("as in freedom") Implementierung der `ssh` Werkzeuge. Die Abkürzung `ssh` steht dabei für *secure shell*, der Name impliziert also schon, dass es primär darum geht eine Shell auf einem nicht-lokalen System zu starten.

ssh:

Wie `cron` besteht auch `ssh` aus einem *Daemon* - dem `sshd` - und einer Familie an Werkzeugen dazu. Der `sshd` wartet üblicherweise auf Verbindungen eines `ssh`-Clients. Das zugehörige `ssh`-Protokoll, das die Komminikation zwischen `ssh` Client und Server regelt, läeuft dabei ueber `TCP`, der `sshd` wartet standardgemäß auf Port 22 auf eingehende Verbindungen.

ssh Client:

Der `ssh` Befehl kann genutzt werden, um sich mit einem `ssh`-Server zu verbinden. Der Name oder die IP-Adresse des Servers *muss* dabei als Argument an den `ssh` Befehl übergeben werden. Der Nutzername, der beim remote System genutzt werden soll, kann dem Host dabei in der Form `name@host` vorangestellt werden.

ssh Client:

Der `ssh` Befehl kann genutzt werden, um sich mit einem `ssh`-Server zu verbinden. Der Name oder die IP-Adresse des Servers *muss* dabei als Argument an den `ssh` Befehl übergeben werden. Der Nutzername, der beim remote System genutzt werden soll, kann dem Host dabei in der Form `name@host` vorangestellt werden.

ssh & Authentifizierung:

Der `sshd` unterstützt eine Vielzahl an Authentifizierungsmethoden, die einfachste davon ist die übliche Anmeldung mit Passwort.

Beispiel:

Als Stundent der Univie hat man 1GB "webspace". Um diesen zu nutzen kann man sich via `ssh` mit seinem *u:account* auf `login.univie.ac.at` einloggen. Dort findet sich im Home-Verzeichnis der Ordner `html`. Auf jede dort abgelegte Datei kann dann via
`homepage.univie.ac.at/<UserID>/pfad/zur/datei`
zugegriffen werden. Achtung: Berechigungen der Dateien nicht vergessen! Der Webserver muss die Dateien lesen können, "others" sollte also Leserechte bei den Dort abgelegten Dateien haben. Wir werden später sehen, wie man Dateien via `ssh` auch kopieren kann.

ssh & pubkeys:

Nachdem passwortbasierte Logins auf öffentlichen Systemen oft das Ziel von Bruteforceangriffen werden empfiehlt es sich als alternative zu Passwörtern bei `ssh` auf *public keys* zu setzen. Dazu generiert ein Nutzer via `ssh-keygen` ein public keypair und legt dann den öffentlichen Teil des Keys (meistens `.pub` Endung) in der `~/.ssh/authorized_keys` Datei am entfernten System ab. Der Login funktioniert nun nicht mehr über das Nutzerpasswort, sondern durch Public Key Authentifizierung.

Bemerkung:

- Um Bruteforce-Angriffe damit wirkungslos zu machen muss der Passwortlogin als Authentifizierungsmethode deaktiviert werden. Dies geschiecht indem man in `/etc/ssh/sshd.conf` den Parameter `PasswordAuthentication` auf `no` setzt.
- Beachte, dass Pubkey Authentifizierung den üblichen Passwort-Login umgeht, also auch mit `usermod -L` gesperrte Nutzer sich via `ssh` einloggen können, wenn die Pubkey Authentifizierung gelingt. Möchte man dies verhindern sollte man für gesperrte Nutzer die login-shell in `/etc/passwd` auf `/sbin/nologin` setzen.

scp:

`ssh` kann auch zur Datenübertragung zwischen Systemen genutzt werden. Hierzu benutzt man den `scp` Befehl. An `scp` wird - analog zu `cp` - eine Liste an Dateien übergeben. Diese können aber ander als bei `cp` auch auf anderen Hosts liegen. Ein Aufruf an `scp` hat also die Form

```
scp [Options] [user@host1:]file1
        ... [user@host2]:file2
```

Achtung: Wie bei `cp` muss auch bei `scp` der `-r` Parameter übergeben werden, wenn Verzeichnisse kopiert werden sollen.

syslog:

`syslog` ist der Loggingservice des Systems und wird von Systemprogrammen (insbesondere also auch den Systemdiensten) dazu genutzt, Log Messages (also Diagnosenachrichten, Infomeldungen, Fehlermeldungen,...) in Logfiles abzulegen. Das Verarbeiten der Logmessages übernimmt der `syslogd`, ein Daemon, der logmessages von verschiedenen Quellen annimmt und diese nach bestimmten Regeln in seperate Dateien schreibt.
Die Konfiguration des `syslogd` findet sich üblicherweise in `/etc/syslog.cong` (oder `/etc/rsyslog.conf`, falls man die `rsyslog` implementierung nutzt). Systemlogs finden sich standardgemäß in `/var/log`

Bemerkung:

Standardgemäß legt `syslogd` zumindest einen Socket `/dev/log` an, an den die meisten lokalen Services ihre Logmessages schicken. Viele `syslogd` Implementationen können auch Nachrichten über das Netzwerk schicken/empfangen, sodass Lognachrichten auf einem zentralen Rechner empfangen und ausgewertet werden können.

Facility:

Lognachrichten werden klassischerweise einer “Einrichtung” (*Facility*) zugeordnet. Diese beschreibt den “Teil” des Systems, der die vorliegende Lognachricht generiert. Mögliche Facilities sind

- auth, authpriv (Sicherheitsrelevante Nachrichten, Authentifizierung)
- cron, daemon (Nachrichten von cron bzw. von Daemonen im Allgemeinen)
- ftp, mail (Nachrichten einen ftp oder mailserver betreffend)
- kern (Lognachrichten des Kernels)
- lpr (Drucker)
- mark (syslog Markierungen)
- news, syslog, user
- uucp (Unix to Unix Copy)

Loglevel:

Innerhalb jeder oben genannten Facility gibt es mehrere Loglevel, die Dringlichkeitsstufen repräsentieren. Standardgemäß sind dies emerg, alert, crit, err, warning, notice, info und debug (vom "wichtigsten" zum "unwichtigsten").

syslog.conf:

Die genaue Sytanx der `syslog.conf` variiert von Implementation zu Implementation, üblich ist es aber immer `syslog`-kompatible Konfigurationen zuzulassen. Dabei hat jede Zeile die Form:

```
facility1.level1;facility2.level2,...    /pfad/zum
```

Bemerkung:

Es ist auch möglich mit , eine Liste von Facilities für ein Level anzugeben, oder mit * alle facilities zu Notieren. Wird ein Level angegeben werden matcht das alle Syslog Nachrichten, die *mindestens* dieses Level haben. Möchte man bei einer `*.level` Regel gewisse facilities ausnehmen, kann man diese mit `;facility.none` angegeben werden.

logger:

Man kann auch der Shell bzw. in Shellscripts Nachrichten an Syslog senden. Dazu verwendet man den `logger`-Befehl. Facility und Level kann mittels `-p facility.level` Parameter angegeben werden. Eine Nachricht muss als Argument übergeben werden.

Festplatten sind oft in mehrere Partitionen unterteilt. Dies hat unter Umständen eine Reihe von Vorteilen, sei es um z.B. System von Nutzerdaten zu trennen, oder durch das Verwenden von `mount`-Parametern wie `nosuid` und `noexec` für mehr Sicherheit zu sorgen. Im wesentlichen gibt es heute zwei mögliche Partitionslayouts: `MBR`-Partitionen und `GPT`-Partitionen.

MBR Paritionen:

Bei MBR Partitionen gibt es 4 *Primärpartitionen*, die Teil des MBR (Master Boot Record) sind. Sie befinden sich damit innerhalb der ersten 512 Bytes einer Festplatte (4*16 Bytes, beginnend bei Byte 446). Um mehr als 4 Partitionen anzulegen kann *eine* Partition als *extended Parition* markiert werden. Auf einer *extended* Partition können dann mehrere *logische* Partition angelegt werden.

fdisk:

Zum erstellen und verändern von MBR Partition kann `fdisk` genutzt werden. `fdisk` wird dabei der Pfad zur Festplatte (ohne Partition) als Argument übergeben. Möchte man nur eine Liste aller vorhandenen Partitionen erhalten kann man `fdisk -l` nutzen.

Befehle in fdisk:

Startet man `fdisk` so bekommt man einen eigenen Prompt, der zur Eingabe von Befehlen an `fdisk` genutzt wird. Folgende Befehle sind unter anderem verfügbar:

- `m`: Gibt eine Liste aller Befehle aus (Help)
- `p`: Gibt eine Liste aller Partitionen aus.
- `d`: Löscht eine Partiton
- `n`: "new", legt eine neue Partition an.
- `t`: "type", legt den Typ einer Partition fest
- `a`: setzt das Boot-Flag auf einer Partition
- `q`: Beenden ohne Änderungen zu schreiben
- `w`: Schreibt geänderte Paritionstabelle und beendet `fdisk`