Gemütlich machen: was beim Login eingestellt werden kann und was das Arbeiten erleichtert

Jetzt, wo wir zumindest in groben Zügen einen Editor beherschen, können wir uns in den Konfigurationsfiles zur bash Einstellungen bei Variablen und Aliasen machen, die das Arbeiten erleichtern können.

Aufgabe: wie hieß nochmal das Konfiguratiosfile der bash?

Variablen setzen beim Login

Bei der bash gibt es eigentlich zwei Files, die ausgeführt werden: .bash_login für Loginshells und .bashrc für andere. Da diese Unterscheidung aber fürs tägliche Arbeiten irrelevant ist, machen wir da keinen Unterschied und schreiben alles, was wir brauchen ins .bashrc. Ins .bash_login schreiben wir nur:

```
source ~/.bashrc
```

damit das .bashrc beim login ausgeführt wird. Selbstverständlich wird vor den eigenen Files das systemweite Configfile ausgeführt, in diesem Fall ist es das /etc/profile (aus historischen Gründen) und /etc/bash.bashrc.

Wir wollen nun Variablen setzen (wie man sie abfragt s.o.!).

Eine Variable wird gesetzt mit

```
NAME=inhalt
```

Sie ist dann aber nur im aktuellen Prozeß (dazu später) verfügbar. Damit sie auch in allen Kindprozessen gilt, brauchen wir hier immer:

```
export NAME=inhalt
```

Dies gilt für die Kommandozeile wie für das .bashrc.

Wichtig: alles, was wir ins .bashrc schreiben, wird erst beim nächsten login ausgeführt. Um das **.bashrc neu einzulesen**, führt man auf der Kommandozeile

```
source .bashrc
```

aus.

Das TMP-Dir

Viele Programme brauchen ein temporäres Verzeichnis, um Dinge ab- oder zwischenspeichern zu können, daher setzen wir zwei Variablen:

```
export TMPDIR=/home/username/tmp
export TMP=/home/username/tmp
```

Stelle sicher, daß das Verzeichnis auch angelegt ist.

Der Prompt / die Eingabeaufforderung

Hier kann man durch geschickte Einstellung sich das Arbeiten sehr erleichtern. Der Prompt wird in der Variable PS1 festgelegt (ja es gibt auch PS2, dies kommt bei einfachen Shellscripten auf der Kommandozeile zum Tragen).

Aufgabe: gib den Inhalt von PS1 aus.

Im Prompt kann man username, hostname, aktuelles Verzeichnis und eine laufende Nummer in der bash-history angeben, daher ist eine sinnvolle Einstellung:

```
export PS1="\u @ \h:\w \!> "
```

Häufig will man sich aber anzeigen lassen, wenn man mit root-Rechten unterwegs ist, daher verwendet man lieber:

```
if [ -w /etc/passwd ]; then
  PS1="\h:\w \!# "
else
```

```
  PS1="\u @ \h:\w \!> "
fi
export PS1
```

dann wird als root das Doppelkreuz # statt > angezeigt.

Aufgabe: schreibe das erste Beispiel auf der Shell und sieh, was sich sofort ändert. Schreibe das zweite Beispiel ins .bashrc. Verwende gerne Kommentare zu Deinen Eintragungen: kennzeichne diese mit vorangestellten # (vor jeder Kommentarzeile).

Nimm gerne weitere Änderungen anhand der Anleitung zur bash vor.

Die Editor-Variable

Gibt den Standard-Editor an. Jede Userin kann diesen für ihre eigene Umgebung selber wählen, aber spätestens als root wird es immer der vi. Daher stellen wir ein:

```
EDITOR=/usr/bin/vi;          export EDITOR
VISUAL=/usr/bin/vi;          export VISUAL
```

Hier neu: mit dem Semikolon kann man mehrere Kommandos in einer Zeile schreiben und trennen.

Die Bash-History

Die bash merkt sich die letzten N Kommandos, wenn man es denn einstellt:

```
HISTSIZE=200;                export HISTSIZE
```

Nach Beenden der Shell werden die Kommandos aus dem Speicher in das File .bash_history geschrieben.

Möchte man nicht, daß ein Kommando in der History landet, schreibt man ein Leerzeichen vor das entsprechende Kommando.

Um Dubletten zu vermeiden, setzen wir außerdem:

```
history_control=ignoredups;              export history_control
set command_oriented_history;            export command_oriented_history;
```

Hacker setzen gern die HISTSIZE=0, damit nicht mitprotokolliert wird, was sie tun.

Aufgabe: da das History File erst beim Beenden der Shell geschrieben wird, was muss ein Hacker genau tun, um das zu verhindern oder seine Spuren zu verwischen?

Die Path-Variable

..hatten wir schon behandelt. Als root möchte man das aktuelle Verzeichnis (".") nicht dabeihaben, deswegen setzen wir:

```
if [ -w /etc/passwd ]; then
   PATH="$HOME/bin:/usr/local/bin:/bin:/usr/bin:/sbin:/usr/sbin:/usr/X11R6/bin"
 else
   PATH="$HOME/bin:/usr/local/bin:/bin:/usr/bin:/sbin:/usr/sbin:/usr/X11R6/bin:."
fi
```

Der Manpath

Ähnlich wie die Pfad-Variable legen wir noch die Pfade fest, wo nach manpages gesucht wird:

```
MANPATH=/usr/man:/usr/local/man:/usr/X11R6/man:/usr/openwin/man:/usr/share/man:/usr/share/catman:/usr/ca
tman;       export MANPATH
```

Aufpassen: alles muss in eine Zeile! Kein Umbruch!

Diverse kleine Dinge

```
export PAGER=less
```

(nur dass das klar ist :))

```
export LC_COLLATE=C
```

bestimmt die Sortierreihenfolge bei ls: zuerst "unsichtbare Dateien". Diese beginnen mit einem Punkt und werden nur bei ls -a angezeigt, sodann alphabetisch sortiert. Erst dann kommen die regulären Dateien, die mit Buchstaben oder Zahlen beginnen in der Reihenfolge: Ziffern, Großbuchstaben, Kleinbuchstaben (dies ist eine Empfehlung).

Sinnvolle Aliase

IdR. gilt: eigene Abkürzungen und Aliase NICHT so nennen wie das Ursprungskommando. Allerdings gibt es Ausnahmen, die auch nicht dramatisch sind: less ist z.B. nur gut nutzbar, wenn der Inhalt der ausgegebenen Datei nicht beim Beenden von less verschwindet.

Daher:

```
alias less='less -MeiQX'
alias les=less
```

ebenso:

```
alias df="df -h"
```

Aufgabe: finde heraus, was die Optionen im Einzelnen bedeuten

Als weiterhin sinnvoll haben sich ergeben:

```
alias 0="sudo $SHELL"
alias    ls='ls -F'
alias    ..='cd ..'
alias pstree='pstree -A -u'
alias rmi="rm -i"
```

Wieder zu allen Kommandos die **Aufgabe:** finde heraus, was sie tun und was die Optionen heißen.

Funktionen

Auch einfache Funktionen können definiert werden:

```
function ll     () { ls -alFh $* | less -MeiQ ;}
function l      () { echo $1* ;}
```

Über Prozesse sprechen wir noch, und auch über grep:

```
function psg    ()
{
  echo 'UID PID PPID C STIME TTY TIME CMD'
  ps -ef | grep $* | grep -v grep
}
```

Der Phantasie sind kaum Grenzen gesetzt und für komplizierte Kommandos kann man sich immer Abkürzungen bauen.

Das .vimrc

Auf älteren Unices benutzt man hier .exrc, also falls Ihr mal das .vimrc nicht vorfindet.

Auch hier sind mannigfaltige Einstellungen möglich! Startet den vi und ruft

```
:set
```

auf, da wird schon eine Menge angezeigt. Ich erachte als sinnvoll:

```
set autoindent       # Einrückungen in der nexten Zeile übernehmen, fürs Programmieren
set wrapmargin=8     # 8 Zeichen vor Zeilenende in die nächste gehen
set textwidth=72     # Text nicht breiter als 72 Zeichen machen, das ist der kleinste gem. Nenner
```

Braucht man einmal Zeilen, die länger sind und weil man Textpassagen mit Einrückungen hereinpasten möchte, setzt man im laufenden Editor:

```
:set paste
```

das überschreibt die Einstellungen mit Zeilenumbrüchen.

Zurückgesetzt wird es mit:

```
:set nopaste
```

Was aber, wenn ich einen Zeilenumbruch habe und den gar nicht will oder der da gar nicht sein darf (s.o.)? Ich gehe in den Bewegungsmodus (ESC) und gehe auf die obere Zeile. Mit J (groß J) füge ich die untere Zeile mit der oberen zusammen.

Revision #1
Created 10 March 2022 15:15:24 by merline
Updated 13 June 2022 11:48:59 by merline