Programmiermethoden und -werkzeuge 1

Woche 6 - Environment

Jochen Hosenfeld

jochen.hosenfeld@informatik.hs-fulda.de

Fachbereich Angewandte Informatik

November 28, 2025

Inhalte des Moduls

Grundlegendes Arbeiten mit der Kommandozeile

Umgang mit Text-Editoren im Terminal

Umgebungsvariablen und Shell-Skripte

Laden von Software-Projekten aus Open-Source-Quellen

Build-Systeme und Paketmanager

Installation und De-Installation von selbst übersetztem Code

Inhalte des Moduls

Grundlegendes Arbeiten mit der Kommandozeile

Umgang mit Text-Editoren im Terminal

Umgebungsvariablen und Shell-Skripte

Laden von Software-Projekten aus Open-Source-Quellen

Build-Systeme und Paketmanager

Installation und De-Installation von selbst übersetztem Code

Functions

- Funktionen in Bash erstellen temporäre Befehle für spätere Aufrufe.

```
1 mcd () {
2     mkdir -p "$1"
3     cd "$1"
4 }
```

```
1 sum () {
2     echo "$1 + $2 = $(($1 + $2))"
3 }
```

Scripts

```
1 read -p "Your name? " name
2 if [[ $name = $USER ]]; then
3     echo "Hello, me."
4 else
5     echo "Hello, $name."
6 fi
```

Was ist ein Bash Script?

- Eine Datei, die einen oder mehrere Befehle enthält
- Bash ist nicht nur eine Kommandozeile, sondern auch ein Interpreter …
- … und kann deshalb diese Dateien einlesen und die Befehle der Reihe nach ausführen.
- Das Arbeiten mit der Kommandozeile kann meistens in Scripts umgewandelt werden und umgekehrt.

Erstellen eines Bash Scripts

1. Schreiben eines Scripts

- Einfache Textdatei mit Texteditor wie Nano und Vim

2. Ausführbarkeit des Scripts erlauben

- Textdateien werden standardmäßig nicht als Programm behandelt

3. Script so ablegen, dass Bash es finden kann

- Bash durchsucht automatisch bestimmte Verzeichnisse nach ausführbaren Dateien

Hello World Script 1

```
1  echo 'Hello World!'
```

Hello World Script 2

```
vim hello_world.sh    # sh als Dateiendung könnte weggelassen werden, ist aber Konvention
```

```
#!/bin/bash
# This is our first script.

echo 'Hello World!'
```

```
ZZ      # in Vim
```

Hello World Script 3

```
johos@AI-HOSENFELD-NB:~/scripts$ bash hello_world.sh
Hello world!
johos@AI-HOSENFELD-NB:~/scripts$ hello_world.sh
hello_world.sh: command not found
johos@AI-HOSENFELD-NB:~/scripts$ ./hello_world.sh
-bash: ./hello_world.sh: Permission denied
```

Hello World Script 4

```
johos@AI-HOSENFELD-NB:~/scripts$ l -l
total 4
-rw-r--r-- 1 johos johos 20 Nov 10 20:34 hello_world.sh
johos@AI-HOSENFELD-NB:~/scripts$ chmod 755 hello_world.sh
johos@AI-HOSENFELD-NB:~/scripts$ l -l
total 4
-rwxr-xr-x 1 johos johos 20 Nov 10 20:34 hello_world.sh*
johos@AI-HOSENFELD-NB:~/scripts$ ./hello_world.sh
Hello world!
```

Berechtigungen

Executable Permissions

- **755**: Üblich, kann vom jedem Benutzer ausgeführt werden
- **700**: Üblich, kann nur vom Eigentümer ausgeführt werden
- **777**: Unüblich, vollständiger Zugriff für alle Nutzer ist oft ein erhebliches Sicherheitsrisiko
- -> Direkte Ausführung (./script.sh): Braucht Leserechte + Ausführrechte
- -> Indirekte Ausführung (bash script.sh): Reichen nur Leserechte

Hello World Script 5

```
johos@AI-HOSENFELD-NB:~/scripts$ hello_world.sh
hello_world.sh: command not found
johos@AI-HOSENFELD-NB:~/scripts$ ls
hello_world.sh
johos@AI-HOSENFELD-NB:~/scripts$ which hello_world.sh
johos@AI-HOSENFELD-NB:~/scripts$ which ./hello_world.sh
./hello_world.sh
johos@AI-HOSENFELD-NB:~/scripts$ which ls
/usr/bin/ls
johos@AI-HOSENFELD-NB:~/scripts$ type -a ls
ls is aliased to `ls --color=auto'
ls is /usr/bin/ls
ls is /bin/ls
```

Linux-Dateisystem (1/2)

Pfad	Abkürzung (Bedeutung/Herleitung)	Beschreibung
/bin	binaries (aus „binary“)	Essentielle Befehlsprogramme für alle Benutzer (auf modernen Systemen oft Symlink zu /usr/bin).
/sbin	system binaries	Essentielle Systemprogramme für Administration (typisch von root genutzt; oft Symlink zu /usr/sbin).
/dev	devices	Gerätedateien, Schnittstellen zu Hardware und Kernel-Subsystemen.
/etc	et cetera	Host-spezifische Systemkonfigurationsdateien.
/home		Benutzerverzeichnisse des Systems.
/lib	libraries	Essentielle Systembibliotheken und Kernel-Module (auf 64-Bit-Systemen oft zusätzlich /lib64; häufig Symlink zu /usr/lib).
/sys	system (sysfs)	Virtuelles sysfs: Informationen und Schnittstellen zur Kernel-/Geräte-Konfiguration.

Linux-Dateisystem (2/2)

Pfad	Abkürzung	Beschreibung
/tmp	temporary	Temporäre Dateien; je nach Systemkonfiguration beim Neustart geleert.
/usr	Unix System Resources, ursprünglich „user“	Teilbare, im Betrieb meist schreibgeschützte Daten: Großteil der Programme, Libraries, usw.
/usr/bin		Nicht-essentielle Befehlsprogramme für alle Benutzer (Hauptort für Benutzerprogramme).
/usr/sbin		Nicht-essentielle Systemverwaltungsprogramme (typisch von root genutzt).
/usr/local/bin		Lokale, nicht distributionsverwaltete Benutzerprogramme (z. B. selbst installiert/kompiliert).
/boot		Dateien zum Systemstart: Kernel, Initramfs/Initrd, Bootloader-Konfigurationen.
/var	variable	Veränderliche Daten wie Logs, Caches, usw.

Unterschiede von Programmtypen

Kategorie	Beschreibung	Speicherort	Beispiele
Befehlsprogramme	Ausführbare Dateien, die über die Kommandozeile für bestimmte Aufgaben aufgerufen werden.	`/bin`, `/usr/bin`, `/usr/local/bin`	`ls`, `cp`, `mv`, `vim`, `gcc`
Systemprogramme	Notwendige Programme für die Systemverwaltung und den Betrieb, oft mit erweiterten Berechtigungen.	`/sbin`, `/usr/sbin`	`shutdown`, `fdisk`, `systemd`
Systemspezifische Konfigurationsdateien	Dateien, die Systemeinstellungen enthalten und die Ausführung von Software und Diensten regulieren.	`/etc`	`fstab`, `passwd`

PATH

```
1 echo $PATH
2
3 # Übliche Pfad für eigene Scripts
4 /usr/local/bin            # Für alle Benutzer zugänglich
5 $HOME/bin                 # Nur für den jeweiligen Benutzer zugänglich
6
7 # Hinzufügen eines Pfades zu der PATH - Variable
8 export PATH="$HOME/bin:$PATH"
```

PATH

PATH ist eine Umgebungsvariable, die die Verzeichnisse auflistet, in denen das System nach ausführbaren Dateien sucht, wenn ein Befehl eingegeben wird.

Best Practice

Was ist, wenn ein Script von einem Python-Interpreter ausgeführt werden soll?

- Hinzufügen einer Codezeile, die angibt, welcher Interpreter für die Skriptausführung verwendet werden soll
- # leitet normalerweise ein Kommentar ein
- In Kombination mit ! ergibt es ein besonderes Zeichen: #!
- *Shebang*, aus Sharp + Bang (Sharp für # und Bang für !)
- Manchmal auch Hashbang genannt
- Besonders nützlich bei Automatisierungen und plattformübergreifenden Anwendungen

Shebang

```
1  #!/usr/bin/python       # gut
```

Shebang

```
1 #!/usr/bin/python       # gut
2 #!/usr/bin/env python   # besser, da höhere Portabilität
```

- Platzierung am Anfang des Scripts
- **#!/usr/bin/env** Nutzt die **PATH**-Umgebungsvariable, um den korrekten Interpreter zu finden

Environment

- Sammlung von Variablen für Shell- und Anwendungsprozesse
- Rolle der Umgebung in einem Linux-System
- Auswirkungen auf Programme und Skripte

  → Konfigurationsdateien

Umgebungsvariablen anzeigen

```
printenv                    # zeigt alle Variablen

printenv | less             # alle Variablen in less 'pipen'

vim
:r !printenv                # alle Variablen in Vim anzeigen

printenv > env_output.txt   # alle Variablen in Datei speichern
```

Umgebungsvariablen anzeigen

```
printenv                    # zeigt alle Variablen

printenv | less             # alle Variablen in less 'pipen'

vim
:r !printenv                # alle Variablen in Vim anzeigen

printenv > env_output.txt   # alle Variablen in Datei speichern

printenv | vim              # alle Variablen in Vim 'pipen'?
```

Umgebungsvariablen anzeigen

```
printenv                    # zeigt alle Variablen

printenv | less             # alle Variablen in less 'pipen'

vim
:r !printenv                # alle Variablen in Vim anzeigen

printenv > env_output.txt   # alle Variablen in Datei speichern

printenv | vim              # alle Variablen in Vim 'pipen'?
printenv | vim -            # alle Variablen in Vim 'pipen'
```

Initialisierung des Environments

Shell-Matrix

	Login-Shell	**Nicht-Login-Shell**
Interaktive Shell	Standard bei direktem Einloggen ins System (z. B. über Konsole oder SSH)	Beim Starten einer neuen Shell innerhalb einer bestehenden Sitzung, üblicherweise zum Testen (z. B. durch `bash` im Terminal)
Nicht-interaktive Shell	Automatisierte Aufgaben, bei denen eine Login-Umgebung benötigt wird	Skripte und Batch-Jobs ohne Benutzereingabe (z. B. `cron`-Jobs)

Login-Shell-Startdateien

Datei	Inhalt
`/etc/profile`	Ein globales Konfigurationsskript, das für alle Benutzer gilt.
`~/.bash_profile`	Persönliche Startdatei eines Benutzers. Kann verwendet werden, um Einstellungen im globalen Konfigurationsskript zu erweitern oder zu überschreiben.
`~/.bash_login`	Falls `~/.bash_profile` nicht gefunden wird, versucht Bash, dieses Skript zu lesen.
`~/.profile`	Falls weder `~/.bash_profile` noch `~/.bash_login` gefunden werden, versucht Bash, diese Datei zu lesen. Dies ist der Standard in Debian-basierten Distributionen wie Ubuntu.

Non-Login-Shell-Startdateien

Datei	Inhalt
`/etc/bash.bashrc`	Ein globales Konfigurationsskript, das für alle Benutzer gilt.
`~/.bashrc`	Persönliche Startdatei eines Benutzers. Kann verwendet werden, um Einstellungen im globalen Konfigurationsskript zu erweitern oder zu überschreiben.

Tip

Zusätzlich zum Lesen der Startdateien lesen Non-Login-Shells die Umgebungsvariablen vom übergeordneten Prozess, normalerweise einer Login-Shell.

.bashrc

- rc: “run command”
  - → Bei Start Initialisierung eines Programms
- Beispiel-Inhalte: Aliases, Funktionen
- Ideal für häufig verwendete Anpassungen

.bashrc

Aus Benutzersicht sehr wichtig, da sie fast immer gelesen wird: Standardmäßig von Nicht-Login-Shells und auch von Login-Shells, da deren Startdateien oft so geschrieben sind, dass sie `~/.bashrc` einlesen.

Dotfiles

- Konfigurationsdateien im Klartext für viele Programme
- Dateinamen beginnen mit einem Punkt (z. B. `~/.bashrc`)
- Standardmäßig in Verzeichnissen verborgen

Organisation von Dotfiles

- Dotfiles in einen eigenen Ordner legen
- Versionskontrolle verwenden
- Mit Symlink verlinken

Symlink

Symlink - Symbolischer Link

Datei, die als Verweis oder Zeiger auf eine andere Datei oder ein anderes Verzeichnis fungiert. Ermöglicht Zugriff auf Dateien/Verzeichnisse an unterschiedlichen Speicherorten unter einem gemeinsamen Namen.

```
ln -s ~/src/dotfiles/vimrc ~/.vimrc     # Beispiel für das Erstellen eines Symlinks

ln                                      # Kommando zum Erstellen eines Links
-s                                      # Option für symbolischen Link

~/src/dotfiles/vimrc                    # Zielpfad der Originaldatei
~/.vimrc                                # Pfad und Name des zu erstellenden Links
```

Vorteile der Organisation

- Einfache Installation
  → Schnelles Setup auf neuen Maschinen
- Portabilität
  → Konsistentes Verhalten der Werkzeuge auf allen Geräten
- Synchronisierung
  → Konfigurationen überall auf dem neuesten Stand
- Änderungsverfolgung
  → Pflege der Versionsgeschichte für langlebige Projekte

Shell Variablen

Merkmal	Umgebungsvariablen	Shell-Variablen
Sichtbarkeit	Global, von Subprozessen vererbt	Lokal zur aktuellen Shell
Setzen	Mit `export VAR=value`	Einfach `VAR=value`
Typische Verwendung	Konfiguration der Systemumgebung	Temporäre Werte für Shell-Skripte oder Sitzungen
Beispielvariablen	`PATH`, `HOME`, `USER`, `SHELL`	`VAR`, `count`, `temp`, benutzerdefinierte Variablen
Gültigkeit	Bleibt in Kindprozessen bestehen	Nur in der aktuellen Sitzung gültig
Speicherung	Dauerhaft in Startup-Dateien oder systemweit	In Skripten oder interaktiven Sitzungen

Häufig genutzte Variablen

Variable	Beschreibung
`BASH_VERSION`	Aktuelle Bash Version als String
`HOSTNAME`	Hostname des Computers
`PWD`	Aktuelles Arbeitsverzeichnis
`RANDOM`	Generiert (pseudo)zufällige Zahl zwischen 0 und 32767
`UID`	ID-Nummer des aktuellen Benutzers
`COLUMNS`	Anzahl der möglichen Zeichen in einer Zeile
`LINES`	Anzahl der möglichen Zeilen des Terminals
`HOME`	Home-Verzeichnis des aktuellen Benutzers
`PATH`	Eine durch Doppelpunkte getrennte Liste von Pfaden, die durchsucht werden, um einen Befehl zu finden.

RANDOM

- Spezielle Shell-Variable
- Generiert (pseudo)zufällige Zahl zwischen 0 und 32767
- Bis zur maximalen Zahl für ein 15-Bit-Integer
- *pseudo random number generator* (PRNG)

Pseudozufall

Partneraufgabe: *Was könnte Pseudozufall bedeuten? Welche Vorteile könnten Pseudozufallszahlen bieten?*F

Pseudozufall

- Für viele Zwecke hinreichend zufällig, aber nicht kryptografisch sicher
- Deterministischer Algorithmus
- Startwert des Algorithmus wird **Seed** genannt

Deterministischer Algorithmus

Für die gleiche Eingabe folgt auch immer die gleiche Ausgabe und zusätzlich wird die gleiche Folge an Zuständen durchlaufen.

Vorteile von Pseudozufallszahlen

- Reproduzierbarkeit
  → Dieselben Ergebnisse in mehreren Durchläufen
- Kontrolle
  → Startzustand oder beim Debugging
- Variabilität
  → Unterschiedliche Seeds führen zu unterschiedlichen Sequenzen

Verwendung von RANDOM

```
1  echo $RANDOM                # Zufallszahl zwischen 0 und 32767
2  echo $(($RANDOM % 10))      # Zufallszahl zwischen 0 und 9 durch Modulo-Operator
3
4  RANDOM=42                   # Setzen des Seeds
5
6  # Zufallszahl in einem benutzerdefinierten Bereich  (z. B. zwischen min und max)
7  min=-100
8  max=100
9  echo $((RANDOM % (max - min + 1) + min))
```

((...)) - Compound Command (Zusammengesetzter Befehl)

Die Klammern `((...))` in Bash ermöglichen die arithmetische Auswertung, und das `$`-Zeichen davor sorgt dafür, dass das Ergebnis dieses Ausdrucks im Befehl verwendet wird.

Zusammenfassung

Environment

PATH

(Einfaches) Script und Shebang

Dotfiles und Symlink

Shell Modus

RANDOM