Tutorial 1: Shell Basics

ASE WS 2024/25

Dieses Dokument ist kein eigenständiges Tutorial sondern unterstützt die Übung. Die grundlegenden Konzepte werden vorgestellt, hilfreiche Links erleichtern die selbständige Vertiefung.

Vorbereitung: wir brauchen eine Shell

Wir brauchen heute für die Beispiele eine simple Linux-Shell und ein paar Linux-Programme.

Option 1: Shell auf dem lokalen Rechner öffnen

Option 2: SSH nach Segfault.net

Anmelden über SSH als Nutzer `root` auf einem 'disposable root server' via segfault.net. Entweder über Putty oder über das `ssh`-Programm (seit kurzem auch unter Windows verfügbar).

```
ssh root@segfault.net # Password is 'segfault'
```

Hinweis: wenn man das Passwort eingibt, werden keine Zeichen angezeigt.

Achtung: Der Server ist nicht sicher - keine vertraulichen Infos/Passwörter dort eintippen

Erster Schritt auf dem Server: Informationen zu Netzwerk-Schnittstellen prophylaktisch in einen Text-Editor kopieren

Zweiter Schritt: tmate installieren und ausführen.

```
apt install tmate
tmate
```

`tmate` startet eine `tmux`-Session und gibt Pfade/Befehle aus, mit denen man der Session remote beitreten kann. Diese Infos am Besten auch in einem Texteditor speichern und 'q' drücken.

Tipp: um Text aus der Shell zu kopieren: Text markieren und dann Rechtsklick (Putty) bzw. `Ctrl-Shift-C` unter Linux/macos.

Grundkonzept: Dateisystem

Das Dateisystem ist *die* zentrale Datenstruktur in UNIX-Systemen. Alle Einstellungen und Daten finden sich dort.

Wichtige Verzeichnisse (Linux):

- / (das Root-Verzeichnis)
- /root/ (das Home-Verzeichnis des Administrator-Accounts 'root')

- /home/user123/ (das Home-Verzeichnis des Nutzers 'user123')
- /etc/ (systemweite Konfigurationen)
- /tmp/ (temporäre Dateien)
- /proc/ (Informationen zu System und Prozessen)
- /sys/ (dynamische Systemkonfiguration / Systemstatus des Kernels)

Wichtig: Dateien und Verzeichnisse mit einem Punkt am Anfang ('dotfiles') werden normalerweise nicht angezeigt oder bei Wildcard-Selektionen ausgewählt. `ls -a` zeigt auch solche unsichtbaren Dateien an.

Hilfreiche Tools I

Finde heraus, was folgende Befehle machen:

- pwd
- cd <directory>
- ls <file/directory/.>
- man <program>
- cat <file>
- mkdir <directory>
- rmdir <directory>
- cp <file/directory (from)> <file/directory (to)>
- mv <file/directory (from)> <file/directory (to)>
- rm <file/directory>
- chmod <file/directory>
- chown <file/directory>

Tipp: `man pwd` zeigt eine Anleitung für `pwd` an. Mit drücken der Taste 'q' beendet man diese wieder.

Aufgaben:

- Erzeuge eine Verzeichnisstruktur in deinem Homeverzeichnis
- Setze die Rechte so, dass niemand außer Dir den Ordner „secret“ lesen kann
- Lösche die Dateien wieder

Grundkonzept: Pipes & Filters

Pipes (dargestellt durch das '|'-Zeichen) erlauben es, die Ausgabe eines Programms als Eingabe für ein anderes Programm zu verwenden. *Filter* sind Programme, die Daten von ihrer Standardeingabe lesen und modifiziert auf der Standardausgabe ausgeben. Damit lassen sich mächtige Verarbeitungspipelines bauen.

Spezielle "Dateien": stdin, stdout, stderr

stdin ist die Standardeingabe, von der ein Programm liest. Daten werden normalerweise auf *stdout* ausgegeben. In einer interaktiven Shell gehen Benutzereingaben an *stdin*; *stdout* wird auf der Konsole ausgegeben. Wenn man Fehlermeldungen oder Statusinformationen ausgeben will, ohne die reguläre

Ausgabe zu unterbrechen, kann man *stderr* verwenden. Standardmäßig wird *stderr* auch auf der Konsole ausgegeben.

>, >>, 2> - Ausgabe in eine Datei schreiben

Beispiel:

- `echo "TEST" > test.txt` - überschreibt den Inhalt von test.txt mit „TEST“
- `echo "HALLO" >> test.txt` - hängt „HALLO“ an das Ende von test.txt an
- `cat *.txt 2> errors.txt` - gibt den Text aller Textdateien auf der Standardausgabe aus und schreibt Fehlermeldungen in die Datei errors.txt

< - Daten aus einer Datei lesen

Beispiel:

- `grep "kuchen" < food.txt` - liest die Datei food.txt auf stdin und sucht darin nach „kuchen“

| - Daten an ein anderes Programm übergeben

Beispiel:

- `cat test.txt | grep "Kuchen" | grep "Baum"` - gibt alle Zeilen von test.txt aus, in denen sowohl „Kuchen“ als auch „Baum“ vorkommen

Hilfreiche Tools II

Finde heraus, was folgende Befehle machen:

- uniq
- cut
- diff
- wc
- grep
- sed
- tr
- sort

Write a Bash *oneliner* which does the following:

- download a text file from http://download.nust.na/pub/gutenberg/etext98/2ws3410.txt if it is not yet present in the current directory (use `wget` for this)
- make all text lowercase (use `tr`)
- split it into individual words per line (use `cat` and/or `sed` for this)
- alphabetically sort the list of words and remove duplicates (`sort` and `uniq`, possibly also `grep`).

- print out the 10 most common words in the text (without number of occurrences) on *stdout* (`uniq`, `sort`, and `head`)

Beispiellösung:

```
cat 2ws3410.txt | sed -e 's/[ ,.!?-]/\n/g' | tr [:lower:] [:upper:] | sort | uniq -c | sort -rn | head -n 22 | tail -n 20 | cut -b 9-
```

Weitere Links:

- Commandline-Tutorial auf PracticalDataScience.org
- Learn Enough Command Line to Be Dangerous
- UNIX game - Pipes & Filters kombinieren um Aufgaben zu lösen.

Terminal Multiplexing

tmux ist eine Art *Window Manager* für die Kommandozeile:

- virtuelle Desktops
  - `Ctrl-B <und danach> c` - neuer Desktop
  - `Ctrl-B p` - vorheriger Desktop
  - `Ctrl-B n` - nächster Desktop
  - `Ctrl-B &` - Desktop schließen (schließt auch Shell und alle darin laufenden Programme)
- Tiling / 'Fenster'
  - `Ctrl-B "` - Fenster vertikal splitten
  - `Ctrl-B %` - Fenster horizontal splitten
  - `Ctrl-B <Pfeiltasten>` - zwischen Fenstern wechseln
- Prozesse im Hintergrund laufen lassen (auch nach SSH-Logout)
  - `Ctrl-B d` - Verbindung zur tmux-Session trennen (detach)
  - `tmux ls` - laufende Sessions anzeigen
  - `tmux attach [name]` - wieder mit einer Session verbinden

→ tmux-Cheatsheet

MacOS

MacOS kennt keine Ctrl-Taste und tmux akzeptiert die Command-Taste nicht. Der einfachste Workaround ist, in der Datei `~/.tmux.conf` den Shortcut auf „Alt-b“ (entspricht Meta-b / M-b) zu ändern:

```
cat > ~/.tmux.conf <<__EOF__
unbind C-b
set -g prefix M-b
bind M-b send-prefix
__EOF__
```

Hilfreiche Tools II

Finde heraus, was folgende Befehle machen (Raphael stellt einige kurz vor):

- tmate
- tmux / screen
- curl / wget
- time
- watch
- file
- grep
- sed
- tr
- sort
- ripgrep / ag / fzf

Test-Dateien z.B.: https://github.com/veltman/clmystery/archive/refs/heads/master.zip (mit wget/curl herunterladen)

Paketmanagement unter Debian/Ubuntu

- `apt show <paketname>`
- `apt search <begriff>`

Befehle, die den Paketindex oder installierte Pakete modifizieren, müssen als Nutzer 'root' oder mit `sudo` ausgeführt werden.

- `apt install <paketname> <paketname>`
- `apt update` (Paketliste aktualisieren)
- `apt upgrade` (alle Pakete aktualisieren)

Weiterführende Methoden:

- andere Repos hinzufügen (siehe `/etc/apt/sources.list/`)
- aus bestimmten Repos installieren (`apt install -t unstable youtube-dl`)
- Pakete manuell installieren (`dpkg -i <Dateiname>`)
- fehlgeschlagene Installation/Konfiguration neu starten (`apt install -f` / `dpkg --configure -a`)

Shell Scripting

Finde heraus, was folgende Bash-Befehle/-Strukturen machen:

- for
- while
- if
- read
- echo

Shell-Scripts sind Textdateien, die Befehle beinhalten, die von der Shell ausgeführt werden. In der ersten Zeile des Scripts steht nach einem speziellen Marker ('Shebang', `#!`) das Programm, das die Befehle in der Datei ausführen sein. Das kann eine normale Shell (z.B. Bash oder Zsh) sein oder auch z.B. ein Python-Interpreter. Da die einzelnen Shells nicht perfekt untereinander kompatibel sind, ist es sinnvoll, Scripte explizit für die Bash-Shell zu schreiben, welche am weitesten verbreitet ist:

```
#!/bin/bash
echo "Hallo Welt"
rm /tmp/test.txt
```

Verbreitete Texteditoren auf der Kommandozeile sind:

- 'nano' (speichern mit `Ctrl-O`, beenden mit `Ctrl-X`)
- 'vim' (speichern mit `<Escape>:w`, beenden mit `<Escape>:q!`)

Siehe auch:

- Bash Cheat Sheet
- Advanced Bash-Scripting Guide

Ausführen von Shell-Scripts

Wenn das Script nicht in einem der Verzeichnis liegt, die für ausführbare Programme verwendet werden (siehe PATH-Umgebungsvariable), muss man den vollen relativen oder absoluten Pfad verwenden ($ steht für den Prompt, nicht für eine Eingabe):

```
$ pwd
/home/test/

$ ls
test.sh

$ test.sh
command not found: test.sh

$ ./test.sh
Hello World!

$ /home/test/test.sh
Hello World!
```

Damit ein Script ausgeführt werden kann, muss auch das 'Executable-Bit' gesetzt sein. Sonst kommt die Fehlermeldung 'Permission Denied'.

```
$ ./test.sh
permission denied: ./test.sh

$ chmod +x test.sh
$ ./test.sh
Hello World
```

Aufgabe

Write the following Bash script:

say_count.sh which does the following:

- check whether the `cowsay` command is available on the system
- if not, download the Debian package from the Debian *unstable* repo and install it
- download the command-line mystery (https://github.com/veltman/clmystery/archive/refs/heads/master.zip) and unpack it into a temporary directory
- find all files which contain the word „Duchess“ and let the cow say how many lines each file has
- clean up, i.e. remove the downloaded files and uninstall `cowsay`

Tipp: Beim Command-Line Murder Mystery trainiert man spielerisch Shell-Befehle und Pipelines.