Inhaltsverzeichnis

1 Einleitung und How to use

Alle kleinen Zusammenfassungen und Code-snippets sind 1:1 unter der originalen Überschrift von Bogner zu finden.

Der Inhalt der Dateien ist von mir teilweise ergänzt oder Teile komplett selbst geschrieben.

Dieses Dokument erhebt keinen Anspruch auf Korrektheit oder Vollständigkeit!

Du kannst entweder über das Inhaltsverzeichnis suchen (klicki-bunti) oder das Stichwortverzeichnis nutzen, um z. B. die Seite zu finden auf der eine if-Anweisung erklärt ist.
Alle Links sind blau.

2 Aufgaben

2.1 bash_ skripting

1.
Erstellen Sie ein Skript, das einen Parameter entgegennimmt und checkt ob es eine Datei gibt, die so heißt wie der übergebene Parameter. Aufruf: . test.sh test.txt Erweiterung: Die Eingabeaufforderung soll solange gezeigt werden, bis der Benutzer ein q eingibt, oder einen vorhandenen Dateinamen.

2.
Programmieren Sie einen Taschenrechner. In einer Eingabeforderung wird der Benutzer aufgefordert eine Zahl einzugeben. Diese Zahl wird zur vorher eingegebenen Zahl hinzuaddiert. Das Ergebnis wird ausgegeben und der Benutzer wird erneut zur Eingabe einer Zahl aufgefordert. Das Endergebnis wird ausgegeben, wenn der User die Zeichenfolge 'exit' eingibt. Die erste Eingabeaufforderung lautet: 'Geben Sie die 1. Zhal ein:'. Die fünfte Eingabeaufforderung lautet:'Bisher wurden 4 Zahlen eingegeben. Das Zwischenergebnis lautet: 22. Geben Sie eine weitere Zahl ein, oder beenden Sie mit exit.' Nach der Eingabe von exit lautet die Konsolenausgabe: 'Endergebnis: 33. Das Programm wurde beendet.' Wird etwas anderes als eine Zahl eingegeben gibt das Programm aus: 'Falsche Eingabe: xxx kann nicht addiert werden. Geben Sie ausschließlich Zahlen ein.'

3.
Speichern Sie das Datum in einer Variablen date.
Speichern Sie den ersten Paramter nach der Eingabeaufforderung in die Variable topic
Erzeugen Sie einen Dateinamen, nach dem folgenden Muster: /home/theo/ <parameter>notes.txt.
Nutzen Sie die Variable topic und die sytemimmanente Variable für das Home-Verzeichnis des aktuellen Users.
Erzeugen Sie die EIngabeaufforderung: "Notiz:" und speichern Sie die Eingabe des Users in der Datei mit Datumsangabe

Die Ergebnisse zu 1 sind hier und hier für 2
Die Lösung für 3 ist hier.
Oder die ultimate Lösung von Bogner zu Nr. 2 hier

3 Dokumentation

3.1 bash_ spezifika

pwd print working dir
whoami als wer bin ich angemeldet

Standarddeskriptoren
IN : Tastatur
OUT : Bildschirm
ERR : Fehler

Bash-Operatoren:
>
das Ergebnis eines Befehls (links von >) wird in eine angegebene Ausgabe geschoben
»
dito, aber Ergebnis wird angehängt
ls etc » datei_ namen.dat
Der BashInterpreter schaut nach, ob es eine Datei <datei_ namen.dat> gibt.
Falls ja, wird das Ergebnis des Befehls <ls> angehängt;
falls nein, wird eine Datei mit dem Namen <datei_ namen.dat> erzeugt
und die Ausgabe des Befehls <ls> reingeschrieben
|
pipe: Das Ergebnis wird an das nächste Programm zur Weiterverarbeitung übergeben
ls -l /etc | less
die Anzeige der Dateien/Ordner vom Verzeichnis etc wird an das Programm less übergeben,
das eine seitenweise Anzeige von Text beherrscht

cat > <dateiname>
Eingabe von beliebigem Text
strg + d beendet die Eingabe und speichert diese in die Datei <dateiname>

Links
Hardlink
ln <dateiname> <dateiname-lnk>
erzeugt eine neue Referenz auf eine Datei in der File-Tabelle der Festplatte.
Jede Datei hat mindestens einen Hardlink.
Softlink
ln -s <verzeichnisname> <verzeichnisname-neu>
Da Hardlinks nur für Dateien möglich sind, gibt es den Softlink für Verzeichnisse.
Es wird eine neue Datei erstellt, deren Inhalt der Verweis auf das Verzeichnis ist

Variablen
Definition:
Variablenname aus Buchstaben, keine Zahlen als 1. Buchstabe, keine Sonderzeichen,
dann Zuweisungsoperator ohne Leerzeichen, dann Wert
var="Zeichenkette"
Verwendung:
$ + Variablenname
echo $var

Kommandosubstitution mit Backticks
var=‘ls‘
speichert in der Variable var die Ausgabe des Befehls

echo $var
gibt das Ergebnis des Befehls aus

3.2 console_ basis

Arbeiten auf der Console

Die Standard-Console unter linux ist die bash (Bourne Again Shell).
Es gibt noch eine Vielzahl von anderen Implementationen. Bekannt und sehr beliebt ist die z-Shell.

Die Console dient der Verwaltung des Systems.
Es gibt drei verschiedene Arten von Befehlen, die mit dem Befehl type spezifiziert werden können

a. Systembefehle die zum Linux-Standardbbefehlssatz zählen
type rm
Ausgabe: Pfad zum Systemprogramm rm: /bin/rm

b. Bashbefehle: Funktionen, die für die Bash geschrieben sind
type cd

c. Alias-Befehle: Befehle mit Optionen, die in der .bashrc definiert wurden
type ls

3.3 linux_ basis

Drei Grundprinzipien von Linux:

- alles ist eine Datei
- Vermeiden von Redundanz: ’sag nur was wenn du was zu sagen hast’: das Ergebnis eines Befehls wird nur im Fall eines fehlers oder wenn das Ergebnis explizit angefordert wird ausgegeben.
- Kleine spezialisierte Programme: ein Programm erledigt nur eine Aufgabe, aber die perfekt

3.4 help

Links:
ubuntuusers
ubuntu_offiziell
wiki von ubuntuuser

Hilfe auf der Console:
man <befehl>
man ls
ls –help

3.5 file_ system

Anzeigen von Dateien und Ordnern

ls
zeigt Inhalt des aktuellen Verzeichnisses

ls -a
zeigt alle Dateien und Ordner des aktuellen Verzeichnisses

ls -l
zeigt Inhalt des aktuellen Verzeichnisses in der Langfassung (Berechtigungen, Datum der Änderungen)

ls Dokumente
zeigt Inhalt des Verzeichnisses Dokumente relativ zum aktuellen Standort.
In diesem Fall befindet sich der Konsolenzeiger im Homeverzeichnis des angemeldeten Benutzers

ls /etc/apache2/
zeigt Dateien und Ordner relativ zur Wurzel (/), d.h. vom absoluten Pfad aus

Bewegen in der Dateihierarchie

- cd Dokumente

  wechselt das Verzeichnis relativ zum Standort

- cd /home/theo/Dokumente

  wechselt mit absoluter Pfadangabe.

  Es ist unwichtig wo der user ist

- cd oder cd

  wechselt in das Home-Verzeichnis des Benutzers

- cd ..

  wechselt eine Ebene tiefer

Manipulieren von Dateien/Verzeichnissen

touch <dateiname>
legt im aktuellen Verzeichnis eine neue Datei an

less <dateiname>
zeigt den Inhalt der Datei an

mkdir <verzeichnis>
legt eine neues Verzeichnis an

cp <quelle> <ziel>
cp <dateiname> <dateiname2>
cp /srv/git/test.txt .
cp /srv/git/test.txt /home/theo/Dokumente
Kopiert eine Datei in das aktuelle Verzeichnies, bzw. das angegebene Verzeichnis

cp -r <quelle> <ziel>
Kopiert ein Verzeichnis

rm <dateiname>
Löscht die Datei

rm -r <verzeichnis>
Löscht das Verzeichnis

mv <dateiname> <dateiname2>
Wird verwendet um eine Datei umzubenennen oder zu verschieben

3.6 find

Wildcard

?

find datei?.txt

das Fragezeichen im Suchbegriff steht für genau 1 Zeichen

*

find d*.txt

der Asterisk steht für beliebig viele Zeichen

[12]

find datei[12].txt

in der eckigen Klammer stehen die Zeichen die gefunden werden sollen

find date[a-k][12]

es ist möglich einen Bereich anzugeben, also zwischen a und k

find datei[!1].txt

alle sollen gefunden werden, die dem Muster entsprechen
und keine 1 als Ziffer nach den Ziffern <datei> haben

Optionen

Es gibt vielfältige Suchoptionen in jedem Berecih.
Es ist möglich nach Dateien zu suchen die eine bestimmte Größe haben,
oder in einem bestimmten Zeitraum verändert wurden, oder einen bestimmten Besitzer haben
Siehe

man find

Beispiel:

find . mmin -100

Findet alle Dateien im akturllen Verzeichnis, die in den letzten 100 Minuten verändert wurden.

3.7 grep

GREP:

grep filtert aus einem Input-Stream Zeilen heraus, die einem Suchmuster entsprechen.
Das Suchmuster wird als RegEx (Regular Expression) übergeben.

Links:

Linwiki
galileo openbook
prontosystem

Beispiel:

lspci | grep VGA

Die Ausgabe des Befehls lspci (Ausgabe der PCI-Komponenten des Systems)
wird über die Pipe an den Befehl grep weitergegeben.
Grep sucht in dem Inputstream nach der Zeichenfolge VGA.
Alle Zeilen die er findet gibt er zum Bildschirm

grep theo /etc/passwd

Der Befehl grep wird aufgerufen.
Der erste Parameter enthält das Suchmuster theo
Der zweite Parameter bzeichnet die Datei (/etc/passwd) in der die Übereinstimmung gefunden werden soll

3.8 vim_ basis_ befehle

********** Befehle zum Moduswechsel**********

- i
  Einfügen vor der Cursorposition
- a
  Einfügen nach der aktuellen Cursorposition
- I
  Einfügen am Anfang der aktuelle Zeile
- A
  Einfügen am Ende der Zeile

*****Befehle zum Löschen / ersetzen von Text ********

- C
  Ersetzt die aktuelle Zeile durch neu eingegebenen Text
- c
  in Verbindung mit Bewegungsoperator ersetzt Text
- ce
  löscht das nächste Wort und fügt ein
- x
  löscht das Zeichen der Cursorposition
- dd
  löscht eine Zeile und kopiert sie in den Cache
- yy
  Zeile kopieren
- v
  Markiermodus starten und mit Pfeiltaste vornehmen
- y
  kopiert den markierten Bereich
- p
  fügt Text aus Zwischenablage nach der aktuellen Zeile wieder ein

****Befehl im Kommandomodus **********

- ESC
  Wechselt vom Einfüge- in den Befehlsmodus
- /
  die eingegebenen Zeichen werden im Text gesucht
- n
  sucht weiter
- N
  sucht rückwärts
- w
  springt ein Wort weiter
- gg
  geht zum Dateianfang
- GG
  geht zum Dateiende
- :w
  speichert die Datei
- :w <dateiname>
  speichert Text in neuer Datei
- :q
  beendet Vim, falls keine Änderungen vorgenommen wurden
- :wq
  Speichert und beendet
- :q!
  verwirft Änderungen und beendet vim

3.9 git_ initial

Funktion:

- git ist eine Versionsverwaltung, ermöglicht also auf frühere Versionsstände einer Datei zuzugreifen
- git wird in der Softwareentwicklung immer verwendet
- git ermöglicht komfortabel das Arbeiten mehrerer Personen an einem Projekt

Kostenfreie Git-Server:
github.com - eigene Projekte können hier abgelegt werden - Projekte anderer Personen stehen hier zur Ansicht zur Verfügung

Erstellen eines lokalen Repositorys:

1. Ordner erstellen
   mkdir EuP
2. Ordner mit git initialisieren
   git init
   #git init EuP erspart den mkdir-Befehl
3. Datei erzeugen zum Testen
   touch test.txt
4. Datei zum Repository hinzufuegen
   git add . -A
5. Datei commiten
   git commit -am "first commit"
6. Das lokale Repos mit dem Remote Repos verbinden
   git remote add origin https://github.com/g16bogner/FIT5H_EuP.git
7. Die lokalen Aenderungen zum Remote Repos hochladen
   git push -u origin master

Aktualisieren eines Projektes

1. mit cd in das Basisverezichnis des Projektes gehen
2. git add . -A
3. git commit -am 'Text zur Charakterisierung'
4. git push origin master

Konfiguration:
A. lOGIN UND pw
git config –global credential.helper Cache
git config credential.helper "Cache –timeout=10000000"

3.10 scripting

Voraussetzungen
SheBang-Zeile:
Die erste Zeile enthält den Pfad zum interpretierenden Programm
#!/bin/bash

Berechtigungen:
Damit eine Datei als Programm ausgefü+hrt werden kann,
muss das Execute_Bit gesetzt sein, d.h. in den Berechrtigungen wird ein x angezeigt.
chmod u+x test.sh

Aufruf:
Es ist eine absolute Pfadangabe notwendig:
./test.sh
Aufruf aus dem aktuellen Ordner

Kontext Variablen
Definition:
trinken='Whisky und Zigarre'
Mehr als ein Wort muss in Anführungszeichen
Zwischen Zuweisung und Variablennamen keine Leerzeichen

Ausgabe:
echo $trinken
Definierte Variablen enthalten zu Beginn ein $ - Zeichen.
Das $-Zeichen bedeutet für den Interpreter generell eine Variable

Variablen - intern
$0 : Der Name der aufgerufenen Datei
$1 : Der Name des 1. Parameters
$# : Anzahl der Skript-Kommandos
$? : Exit-Status des letzten Befehls; wurde der letzte Befehl erfolgreich ausgeführt.

Kommandosubstitution
echo ‘date‘
echo $(date)
Der Shell-Befehl date wird ausgeführt und anschließend an das Programm echo übergeben

Kommandos in Variablen
remove='rm test.txt'
$remove

String-Konkatenierung
string_1='Heia und '
string_2=Popeia
string="$string_1 $string_2 was raschelt im Stroh?"
echo "$string_1 $string_2 aber im Heu"
echo $string

IF Anweisung

```
read -p "Ihre Eingabe: " var_name
if [[ $var_name = "q" ]]; then
        echo "Ist ein prima q"
elif [[ $var_name = "m" ]]
then
        echo "Jo, ein m"
else
        echo "irgendwas anderes"
fi
```

Schleifen:
Die Schleife wird ausgeführt solange bis der Wert der Variable $eingabe nicht 1 ist.

```
while [[ true ]]
do
        read -p "Eingabe: " eingabe
        echo $eingabe
done
```

Wiederholung bis der geprüfte Wert ein false ergibt

```
until [[ false ]]
do
        read -p "Eingabe: " eingabe
        echo $eingabe
done
```

Operatoren
[[$var]] true falls in $var etwas steht außer false oder 0
[[!$var]] true falls $var ist 0, false, oder leer

Strings
[[wert1 = 'string']] gibt true wenn wert1 gleich Zeichenkette ist
[[wert1 != 'string']] gitt true wenn wert1 ungleich Zeichenkette ist

arithmetrische Werte
[[wert1 -eq wert2]] gibt true wenn wert1 gleich wert2
[[wert1 -ne wert2]] gibt true wenn wert1 ungleich wert2
[[wert1 -gt wert2]] gibt true wenn wert1 größer wert2
[[wert1 -lt wert2]] gibt true wenn wert1 kleiner wert2

Systemoperationen
[[-e $dateiname]] gibt true wenn Datei vorhanden ist
[[-d $verzeichnis]] gibt true wenn Verzeichnis vorhanden

4 Aus dem Hauptverzeichnis

4.1 datum.sh

```
#!/bin/bash

test_datum=‘date +%d.%m.%Y‘
echo $test_datum

heute=‘date +%s‘
echo $heute

datum_str="2017-01-01"
datum=‘date -d "${datum_str}" "+%s"‘
echo $datum
```

4.2 schleife.sh

```
#!/bin/bash

while [[ $eingabe -ne 1 ]]
do
   read -p "Mach Eingabe du: " eingabe
   echo $eingabe
done
```

4.3 zeigen.sh

*

```
#!/bin/bash

read -p "Geben Sie die 1. Zahl ein:" zahl1

if [[ ‘echo "$zahl1" | grep [[:digit:]]‘ ]]
   then
   echo "Die 1. Zahl ist eine $zahl1"

else
   echo "Falsche Eingabe"
fi

ergebnis=zahl1
zaehler=1

read -p "Geben Sie eine weitere Zahl ein, oder beenden Sie mit exit: " eingabe

while [[ $eingabe -ne "exit" ]]
do
   ((zaehler ++))
   ergebnis=$((ergebnis+eingabe))
   echo "Bisher wurden $zaehler Zahlen eingegeben. Das Zwischenergebnis lautet: $ergebnis."

   read -p "Geben Sie eine weitere Zahl ein, oder beenden Sie mit exit. " eingabe

   if [[ ! ‘echo "$eingabe" | grep [[:digit:]]‘ ]]
   then
      echo "Falsche Eingabe!"
   fi
done


   echo "Das Endergebnis lautet: $ergebnis"
```

5 Lösungen

5.1 bash_ skripting

1

```
#!/bin/bash
if [[ $1 ]]; then
   abfrage=$1
else
   #read -p "Ihre Eingabe: " abfrage
   abfrage=''
   #while [[ $abfrage != 'q' ]] && [[ ! -e $abfrage ]]
   until [[ $abfrage = 'q' ]] || [[ -e $abfrage ]]
   do
      read -p "Ihre Eingabe: " abfrage
      echo "Die Eingabe lautet: $abfrage"
   done


fi

if [[ -e $abfrage ]]
then
   echo "Der Eintrag ist vorhanden: $abfrage"
else
   echo "Die Datei ist nicht vorhanden!"
fi
```

2

```
#!/bin/bash
zahl=0
counter=1
while [[ 1 ]]
do
   read -p "Ihre $counter . Eingabe: " val

   if [[ $val = "exit" ]]; then
      echo "Endergebnis: $zahl. Das Programm wurde beendet."
      break
   elif [[ $val -gt 0 ]]; then
      counter=$((counter + 1))
      zahl=$((zahl + val))
      echo $zahl
   else
      echo "Falsche Eingabe: $val kann nicht addiert werden. Geben Sie ausschließlich Zahlen ein."
   fi
done
```

3 von Valid:

```
#!/bin/bash
if [[ $1 ]]
   then
   datum=$(date)
   read -p "Notiz: " notiz
   cd
   topic=$1
   if [[ -e $topic ]]
      then
      cd $topic/
      if [[ -e "notes.txt" ]]
         then
         echo $datum >> notes.txt
         echo $notiz >> notes.txt
      else
         touch "notes.txt"
         echo $datum >> notes.txt
         echo $notiz >> notes.txt
      fi
   else
      mkdir $topic
      cd $topic/
      touch "notes.txt"
   fi
else
   echo "Sie haben keinen Ziel-Ordner angegeben."
fi
```

Oder man nehme die gemeinsam im Unterricht erarbeitete Lösung zu 3:

```
#!/bin/bash
if [[ -z $1 ]] #erster Parameter es wird geprüft ob ein Paramter übergeben wurde
   then
   echo "Sie haben keinen Ziel-Ordner angeben."
else
   topic=$1
   datum=$(date +"%d.%m.%Y %H:%M")
   read -p "Notiz: " notiz
   dir="$HOME/$topic"
   file="$dir/notes.txt"
   mkdir -p $dir
   printf "$datum: $notiz \n" >> $file
fi
```

Stichwortverzeichnis