1. Remarks on UNIX

UNIX quotes and pipes

```
"..."       system variables are expanded, but anything else is not expanded
            in the shell
              ls a* --> will show all files starting with an a
              ls "a*" --> will show file a* (thus, the * is not expanded)
              echo "* $path" --> in csh/tcsh: * is not expanded, but $path is
              ecoh "* $PATH" --> in sh/bash: * is not expanded, but $PATH is

'...'       whatever is between ' ' is not expanded in the shell
              ls 'a*' --> will show file a*

`..`        whatever is between ` ` is first executed
              ls `which more` --> will execute which more and pipe the output to ls

>           put output to file (overwriting existing file)
              ls > out

>>          concatenate output to file (appending to exisiting file)
              ls >> out

|           pipe; stream output to next command
              ls | less

.           current directory

..          one directory up

~           home directory

man <command>     help page for command (very useful for sh and awk!)
```

Text commands

```
grep expr file          show line in file that matches expr
                        echo "hi" > out;echo "hello" >> out; grep hi out

grep -i expr file       show line in file that matches expr regardless of case
                        echo "Hi" > out;echo "Hello" >> out; grep -i hi out

sed 's/expr/newexpr/g' file   substitute all occurences of expr by newexpr in file
                              echo "hi" > out; sed 's/hi/hello/g' out
```

Regular Expressions (used in grep, sed, awk, etc)

```
.           any character, just one of them
*           any character, any number of them
[ ]         selection of characters
-           range in selection
^           beginning of line
$           end of line
```

Examples:

```
      echo "hi there Earthlings" | sed 's/h/H/g'
      echo "hi there Earthlings" | sed 's/^h/H/g'
      echo "hi there Earthlings" | sed 's/h./H/g'
      echo "hi there Earthlings" | sed 's/[a-k]//g'
      echo "hi there Earthlings" | sed 's/[eE]/ /g'
      echo "hi there Earthlings" | sed 's/.*//g'
```

Literal

To indicate the literal use of a character, rather than have it interpreted as a regular expression, use \.

Examples (in csh/tcsh):

```
      echo $path
      echo \$path
```

(or, in sh/bash: echo $PATH; echo \$PATH)

UNIX file names

Given the special meaning of certain characters in UNIX, it's highly recommended not to use any of the following characters in file or directory names: "*", " " (a space), "(", ")", "[", "]", " " (a tab), "$", "^", and "\".

A good substitute for a space in a file name is the underscore; for example "ls my_file".

Assignment 1: Create a file with the following content:

```
     DYNA DYN: Step         Time      TOTEner       TOTKe       ENERgy  TEMPerature
     DYNA PROP:             GRMS      HFCTote       HFCKe       EHFCor       VIRKe
     DYNA INTERN:          BONDs       ANGLes      UREY-b    DIHEdrals    IMPRopers
     DYNA EXTERN:        VDWaals         ELEC      HBONds          ASP         USER
     DYNA PRESS:            VIRE         VIRI      PRESSE      PRESSI       VOLUme
     DYNA ACE1:      HYDRophobic         SELF   SCREENing      COULomb
     DYNA ACE2:        SOLVation  INTERaction
      ----------       ---------    ---------    ---------    ---------    ---------
     DYNA>        0      0.00000    -49.09193      0.00000    -49.09193      0.00000
     DYNA PROP>         10.28069    -49.07173      0.06062      0.02021      2.59160
     DYNA INTERN>        0.19551      2.04452      0.00000      0.01765      0.17504
     DYNA EXTERN>        0.99339    -56.31113      0.00000      0.00000      0.00000
     DYNA PRESS>         0.00000     -1.72774      0.00000      0.00000      0.00000
     DYNA   ACE1>        3.79308   -124.87746    113.08680    -44.52047
     DYNA   ACE2>      -11.79066     68.56633
      ----------       ---------    ---------    ---------    ---------    ---------
      UPDECI: Nonbond update at step       105
      UPDECI: Nonbond update at step       126
      UPDECI: Nonbond update at step       166
     UPDECI: Nonbond update at step       488
     DYNA>      500      1.00000    -29.10387      8.79504    -37.89891    260.34504
     DYNA PROP>         26.69956    -29.09088      9.19331      0.01299    -32.42548
     DYNA INTERN>        3.36307      7.50322      0.00000      1.64540      0.27446
     DYNA EXTERN>        2.38945    -56.86189      0.00000      0.00000      0.00000
     DYNA PRESS>         0.00000     21.61699      0.00000      0.00000      0.00000
     DYNA   ACE1>        3.78739   -124.41949    111.74533    -44.18774
     DYNA   ACE2>      -12.67415     67.55760
      ----------       ---------    ---------    ---------    ---------    ---------
```

Then use the grep command to extract all lines containing DYNA> from the file.

2. Scripting

Sh scripting

In the following, we will ------beg to indicate the beginning of the script and -------end to indicate the end of the script. These lines are not part of the script. Create the scripts with the indicated name using an editor (you can copy and paste from this document). Then execute the given commands.

Script do1:

```
-------beg
#!/bin/sh          # always start with this line

                   # white space is ignored

#                  # comments are preceded by #
ls                 # any unix command can be given
-------end

chmod +x do1       to make do1 executable
./do1              to execute do1 (or, if . is in your path, simply type do1).
```

Variables in sh:

```
$#          number of arguments
$*          all arguments
$1          first argument
$2          second argument

shift       move the arguments to the left, decreasing $#
```

Script do2:

```
-------beg
#!/bin/sh
echo "first argument is $1"
echo "second argument is $2"
echo "total number of arguments is $#"
shift
echo "first argument is $1"
echo "second argument is $2"
echo "total number of arguments is $#"
-------end

./do2
./do2 1 2 3 4 5 6
```

If command (with test statements):

```
if [ expr ]
then

fi

if [ expr ]
then

else

fi
```

Examples:

```
      if [ -e do1 ]                   # -e if file exists (directory or file)
      then
         echo "do1 is present"
      else
         echo "do1 is not around"
      fi

      if [ -s do1 ]                # -s if file exists and has more than 0 bytes
      if [ -d .. ]                 # -d for directories
      if [ ! -e do1 ]              # ! means not
      if [ $2 = $1 ]               # if 2 strings are equal
      if [ $2 != $1 ]              # if 2 strings are unequal
      if [ $1 -eq $2 ]     # if the 1st integer is numerically equal to the 2nd
      if [ $1 -ne $2 ]     # if the 1st integer is numerically unequal to the 2nd
      if [ $1 -lt $2 ]     # if the 1st integer is numerically smaller than the 2nd
      if [ $1 -gt $2 ]     # if the 1st integer is numerically larger than the 2nd

      man test             # man page for test
```

Script do3:

```
-----beg
#!/bin/sh

if [ $# -ne 3 ]
then
  echo "usage: do3 file1 num1 num2"
  exit         # exit: quit the script
fi

if [ -e $1 ]
then
  echo "$1 exists"
else
  echo "$1 does not exist"
fi

if [ $2 -gt $3 ]
then
  echo "$2 > $3"
else
  if [ $2 -lt $3 ]    # it's straightforward to nest if statements
  then         # continued on next page
```

```
    echo "$2 < $3"   # continued from previous page
  else
    echo "$2 = $3"
  fi
fi
------end

./do3
./do3 do1 a b
./do3 do1 3 3.4
./do3 do1 3 4
```

Assignment 2: write a script "doa2" that tests for all of its arguments if a directory with the name of that argument is present. For example:

```
      %./doa2 bin tmp lala /usr/bin
        the directory bin is present
        the directory tmp is not present
        the directory lala is not present
        the directory /usr/bin is present
```

Variables:

```
a=$1    # set variable a equal to the first argument
b=$#    # set variable b equal to the number of arguments
c=`expr 1 + 3`   # set variable c equal to the output of the expr 1 + 3 command
```

Script do4:

```
------beg
#!/bin/sh

a="expr $1 + $2"
echo "a= $a"
b=`expr $1 + $2`
echo "b= $b"
c=`\ls d*`
echo "c= $c"
------end

./do4 1 2
```

Looping:

```
for b in $a
do
  ...
done

while [ expr ]
do
  ...
done

break             # exit loop
```

Script do5:
```
------beg
#!/bin/sh
i=0
a=`\ls [a-zA-Z]*[0-9]`
for b in $a
do
  if [ ! -d $b ]
  then
    i=`expr $i + 1`
    echo "file $b is file number $i"
  fi
done
-------end

./do5
```

Assignment 3: write a script that calculates N! where N is given as argument. Use a while loop.
Assignment 4: explain why writing such a script would be hard when using a for loop.
Assignment 5: describe what the following script (do6) does. Also, why do we use \ls instead of ls?

Script do6:
```
-----beg
#!/bin/sh
a=`\ls io/*out`
for b in $a
do
  c=`basename $b .out`
  gzip $b
  gzip dcd/$b.dcd
  gzip data/$b.cor data/$b.rst
done
-----end
```

3. awk

use man awk for a complete overview of awk.

Command line:

```
awk 'awk_commands' file          # when given on the command line, awk commands are
                                 # between ''

awk '{print}'     # print entire line
awk '{print $1}'  # print first column of each line
awk '{print NF}'  # print number of columns of each line
awk '{print $NF}' # print last column of each line
awk '{print $0}'  # print entire line for each line

awk '/hi/{print $1}'         # print 1st column when string hi is encountered
awk '/^[a-zA-Z]/{print $2}'  # print 2nd column when regular expression ^[a-zA-Z] is
                             # encountered
```

Example:
```
echo "hi hello" > out;echo "hello hi" >> out;awk '/^hi/{print $2}' out
```

Assignment 6: use awk to print the time step and the temperature from a CHARMM output file.

File:

```
awk -f awkfile file             # where all awk commands are collected in awkfile
```

Example:
```
echo '{print $1}' > awkfile; awk -f awkfile do1
```

Assignment 7: look at awkfile; why are there no single quotes around the {}?
Assignment 8: explain if the following command works or not:
```
echo "{print $1}" > awkfile; awk -f awkfile do1
```

Within sh:

It's usually the easiest and most versatile to put awk commands inside a script. Multiple awk commands can be separated by a semi-column (;). It is very important that if you want to spread commands out over more than 1 line, that you use \ as the **very last** character of the line. If there are characters past the \ (even a space), errors will occur.

Script do7:
```
-----beg
#!/bin/sh

echo $* | awk '/hello/{print "hello to you too"};\
     /so/{print "yes right"}'
-----end

./do7 "hello computer, please talk back sometime soon"
```

Assignment 9: place a space behind the \ and execute the script.
Assignment 10: Write a script "dot" in which you pass the name of a charmm output file as first argument. Then, the script echos the timestep and the temperature from the charmm output file.

Example:
```
%./dot io/md.out
0 301.45097
500 303.76319
1000 298.12077
...
```

Variables:

These can be introduced at any point. What they contain (string, integer, float) is automatically determined from the context. Arrays can be introduced by using []. Calculations can be performed on variables, and standard math transformations are available.

Examples:
```
      echo "1 2 3" | awk '{x[1]=$1;x[2]=$2;x[3]=$3}'
      echo "1 2 3 4" | awk '{x[1,1]=$1;x[1,2]=$2;x[2,1]=$3;x[2,2]=$4}'
```

Assignment 11: Explain what x[1,2] is.

Script do8:

```
-------beg
#!/bin/sh

echo $* | awk '{x=$1;y=$2;print x+y;print sin(x);print log(y);print exp(-y);}'
-------end

./do8 1 2
```

Assignment 12: is the argument for sin in degrees or radian? What's the base of log?
Assignment 13: write a script "dotc" that takes a charmm output file as first argument. The script should print the time step and the temperature in degrees Celcius.

```
      Example:
      %./dotc io/md.out
      0 29.30097
      500 31.61319
      1000 25.97077
      ...
```

If statements:

```
if (statement){};            single if statement
if (statement){} else {};    notice how the first ; is missing when using {}

Examples:
  echo 1 2 | awk '{x=$1;y=$2;if(x<y)print "x<y";else print "y<x"}'
  echo 1 2 3 | awk '{z=$1;if(($1<$3)||($1<$2))z=$2;print z}'
  echo 1 2 3 | awk '{z=$1;if(($1<$3)&&($1<$2))z=$2;print z}'
```

Assignment 14: what do the || and && mean in the examples above?
Assignment 15: write a script "dotcpos" that takes a charmm output file as first argument. The script should print the time step for which the temperature was larger than 300K.

```
      Example:
      %./dot io/md.out
      0
      500
      3500
      ...
```

Notice that an infinite amount of conditions can be combined using ().

C-like syntax:

```
x++         increase x by 1
x+=1        increase x by 1
x/=6        divide x by 6
x*=2        multiply x by 2
x--         decrease x by 1
x-=7        decrease x by 7
mod(x,3)    remainder of division of x by 3
```

Looping:

```
for(start; until; change){ command }

while(condition){ command }
```

Examples:

```
      echo "1 2 34 5" | awk '{for(i=1;i<=NF;i++)print $i}'
      echo "1 2 34 5" | awk '{i=1;while(i<=NF){print $i;i+=2;}}'
      echo "1 2 34 5" | awk '{i=2;while(i<=NF){print $(i-1);i++;}}'
```

Assignment 16: what is $i in the examples above?
Assignment 17: why do we have two }} at the end of the second example?
Assignment 18: what does the $(i-1) mean in the last example?
Assignment 19: write a script "doesum" that takes a charmm output file as first argument. The script should print the time step and the sum of the TOTKE and ENER fields.

```
      Example:
      %./doesum io/md.out
      0 -49.09193
      500 -29.10387
      1000 -31.28715
      ...
```

Assignment 20: now write a script "doangl" that prints the time step and the sum of the ANGL and DIHE fields of a given charmm output file. Hint: use a variable for the time step.

```
Example:
      %./doangl io/md.out
      0 7.58172
      500 6.59832
      1000 8.14710
      ...
```

String operations:

```
index(a,b)            character position of string a in string b
substr(a,start,tot)   substring of a starting at start with total of tot characters
gsub(a,b,c)           replace string a by b in variable c (a can be regular expr.)
x a b                 concatenate x, a, and b (non-strings will be converted)
tolower(a)            converts a string to lower case
toupper(a)            converts a string to upper case
```

Script do9:
```
-----beg
#!/bin/sh

a=`\ls do*`
for b in $a
do
  echo $b | awk '{gsub("[0-9]","",$0);print}'
done
-----end
```

Script do10:
```
------beg
#!/bin/sh

touch a.pdb bc.pdb
a=`\ls`
for b in $a
do
  echo $b | awk '{i=index($1,".pdb");if(i!=0){x=substr($1,1,i-1) ".cor";print x}}'
done
------end
```

Assignment 21: explain what script do10 does
Assignment 22: write a script "dofind" which takes 1 argument. The script will then see if a file with either the lowercase argument or the uppercase argument exists. Hint: use ` ` to assign awk output to a shell variable. Also, use "touch" to create (empty) files, e.g. "touch TESTFILE".

Example:
```
%./dofind testfile
testfile is not present
TESTFILE is present
%./dofind otherfile
otherfile is present
OTHERFILE is not present
```

BEGIN, END, functions, and comments:

```
BEGIN{}                 is executed before reading the file
END{}                   is executed after reading the entire file
function funct(void){}  introduces a function
#                       is treated as comment
```

Example:
```
echo "1 2 3 4" | awk 'BEGIN{n=0};{for(i=1;i<=NF;i++)n+=$i;};END{print n}'
```

Assignment 23: write a script "dodyna" that counts how many times DYNA> is printed in a given charmm output file.

Example:
```
%./dodyna io/md.out
1000
```

Assignment 24: write a script "doavtemp" that prints the average temperature of a given charmm output file.

```
      Example:
      %./doavtemp io/md.out
      299.36
```

Assignment 25: edit the script "doavtemp" in order to also print out the root mean square fluctuation of the temperature.

```
      Example:
      %./doavtemp io/md.out
      299.36 3.18
```

Script do11:

```
------beg
#!/bin/sh

a=`\ls`
for b in $a
do
  echo $b |\
     awk 'function prt(x){\
            # we do not want to modify x itself
            y=x;\
            gsub(".pdb",".cor",y);\
            # it is recommended that functions always return something
            return(y);};\
          /.pdb/{print prt($1)}'
done
------end
```

Notice that we can define multiple functions within a single awk script.

Assignment 26: write a script "dotdev" that gives the largest observed temperature deviation from 300K in a given charmm output file. In your awk script, introduce the function abs which returns the absolute value of a number. Use this function to determine the larges deviation.

```
      Example:
      %./dotdev io/md.out
      13.45163
```

Input/output, system commands:

```
getline<file          read line from file
close(file)           close file
>file                 write to file (overwriting existing file)
>>file                append to file
system(string)        execute unix command(s) string
```

Beware not to open too many files! This will give rise to untracable errors. Thus: close files after reading.

Beware to open existing files! Otherwise this will give rise to untracable errors.

Passing variables to awk:

```
awk -v var=value                      sets variable var to value
awk -v var1=value1 -v var2=value2     sets both var1 and var2
```

This is incredibly useful: we can now pass values from the sh script to awk!

Script do12:

```
------beg
#!/bin/sh

if [ $# -eq 2 ]
then
  awk -v f1=$1 -v f2=$2 \
    'BEGIN{\
       while((getline<f1)!=0){if($1=="DYNA>") print f1, $0;};\
       close(f1);\
       while((getline<f2)!=0){if($1=="DYNA>") print f2, $0;};\
       close(f2);\
       exit;}'
fi
------end

./do12 io/md1.out io/md2.out
```

Script do13:

```
------beg
#!/bin/sh

if [ $# -eq 2 ]
then
  awk -v f1=$1 -v f2=$2 \
    'BEGIN{\
       n1=0;while((getline<f1)!=0){if($1=="DYNA>") n1++;};\
       close(f1);\
       n2=0;\
       while((getline<f2)!=0){if($1=="DYNA>") n2++;};\
       close(f2);\
       print "file ",f1," had ",n1," while file ",f2," had ",n2;\
       exit;}'
fi
------end

./do13 io/md1.out io/md2.out
```

Assignment 27: write 2 files, each consisting of a single column of numbers. The files should have equal lengths. Then, write a script "dodif" that uses a single awk command to process both files. The script should list the difference between the two numbers; the script should work on any file length.

```
Example:
%cat 1.dat
1
2
3
%cat 2.dat
5
7
9
%./dodif 1.dat 2.dat
-4
-5
-6
%./dodif 2.dat 1.dat
4
5
6
```

Assignment 28: Modify "dodif" in such a way that it will calculate the average difference, and the rmsd in difference.

```
Example:
%cat 1.dat
1
2
3
%cat 2.dat
5
7
9
%./dodif 1.dat 2.dat
-5 1
```

Assignment 29: write a file that consists of a single column of numbers. These numbers should correspond to a selection of time steps of a charmm output file. Then, write script "dot2" that uses awk to extract the temperature of only the steps present in the first file.

```
Example:
%cat steps.dat
500
1500
4500
%./dot2 steps.dat io/md.out
299.14561
301.12119
298.70081
```

More complex examples.

Download the pdb files 1ERK.pdb and 2ERK.pdb; put them in your current working directory.

Script do14:

```
-------beg
#!/bin/sh

if [ $# -ne 1 ]
then
  echo "usage: do14 pdb file"
  exit
fi

pdb=$1
if [ ! -e $pdb ]
then
  pdb="$pdb".pdb
  if [ ! -e $pdb ]
  then
    echo "error: $1 or $pdb not found"
    exit
  fi
fi

awk 'BEGIN{first=1;};\
    {if(($1=="ATOM")||($1=="HETATM")){\
      resn=substr($0,18,3);\
      ires=substr($0,23,4);\
      chain=substr($0,22,1);\
      if(first==1){\
        first=0;\
        print resn,ires,chain;\
        old=resn "-" ires "-" chain;}\
      else{\
        new= resn "-" ires "-" chain;\
        if(old!=new){\
          print resn,ires,chain;\
          old=new;};
      }}}' $pdb
-------end
```

Assignment 30: Explain what script do14 does.
Assignment 31: What does the variable first do, why is it used?
Assignment 32: What is the function of the variables old and new?
Assignment 33: Why are old and new defined as resn "-" ires "-" chain ?

Script do15:

```
-----beg
#!/bin/sh

if [ $# -ne 1 ]
then
  echo "missing residue number"
  exit
fi

awk -v f1=1ERK.pdb -v f2=2ERK.pdb -v ires=$1 \
  'BEGIN{n1=0;\
         # use the !=0 construction below so that getline stops when file is
         # at end (which would return 0)
         while((getline<f1)!=0){\
          if((($1=="ATOM")||($1=="HETATM"))&&((substr($0,23,4)+0)==ires)){\
            n1++;\
            x1[n1]=substr($0,31,8);\
            y1[n1]=substr($0,39,8);\
            z1[n1]=substr($0,47,8);\
         };};\
         close(f1);\
         # now the 2nd file
         n2=0;\
         while((getline<f2)!=0){\
          if((($1=="ATOM")||($1=="HETATM"))&&((substr($0,23,4)+0)==ires)){\
            n2++;\
            x2[n2]=substr($0,31,8);\
            y2[n2]=substr($0,39,8);\
            z2[n2]=substr($0,47,8);\
         };};\
         close(f2);\
         if(n1!=n2)print "error: not same atoms in ",f1," and ",f2;\
         else {\
           rms=0;\
           for(i=1;i<=n1;i++)\
             rms+=( ((x1[i]-x2[i])^2) + ((y1[i]-y2[i])^2) + ((z1[i]-z2[i])^2) );\
           rms=sqrt(rms/n1);\
           print "the rmsd of residue ",ires," in ",f1, " and ",f2," is ",rms;};\
         # we are done
         exit;}'
----end
```

Assignment 34: What does script do15 do?
Assignment 35: What does the -v ires=$1 mean; what is $1?
Assignment 36: Why is there only a BEGIN and no body or END?
Assignment 37: Explain what the "if((($1=="ATOM")||($1=="HETATM"))&& ..." statement does; when will this be true?
Assignment 38: Why do we use the +0 in the if statement (substr($0,23,4)+0)==ires)
Assignment 39: What does the ^ int the "rms+=" line mean?
Assignment 40: What is the value for residue 6. Does this value make sense?
Assignment 41: In do15, we made the assumption that atoms occur in exactly the same order in the 2 files. However, this is not necessarily the case. For example, in the first file we could have atoms "N CA CB C", in the second file we could have atoms "N C CA CB". Modify do15 in such a way that the order of atoms does not matter, that the script always compares CA to CA, CB to CB, etc. Test it with an edited 2ERK.pdb file, in which you switch the order of atoms.

Script do16:

```
-----beg
#!/bin/sh

touch do16.tmp
\rm -f do16.tmp

awk 'BEGIN{n=0;};\
     {if(($1=="ATOM")||($1=="HETATM"))&&($3=="CA")){\
        n++;res[n]=$4 $6;x[n,1]=$7;x[n,2]=$8;x[n,3]=$9;};\
     };\
     END{for(k=1;k<=3;k++){xm[k]=x[1,k];xp[k]=xm[k];};\
         for(i=2;i<=n;i++)for(k=1;k<=3;k++){\
            if(x[i,k]<xm[k])xm[k]=x[i,k];\
            else if(x[i,k]>xp[k])xp[k]=x[i,k];};\
         #
         for(k=1;k<=3;k++)dx[k]=0.5*(xp[k]+xm[k]);\
         #
         for(i=1;i<=n;i++){\
           r=0;for(k=1;k<=3;k++)r+=(x[i,k]-dx[k])^2;\
           print res[i],sqrt(r)>>"do16.tmp";};\
         #
         command="sort -g -k 2 do16.tmp > do16.tmptmp";\
         system(command);\
         n=0;\
         while((getline<"do16.tmptmp")!=0){\
           n++;\
           printf("%s r=%7.2f\n",$1,$2);};}' 1ERK.pdb
-----end
```

Assignment 42: What does do16 do?
Assignment 43: Why do we first touch and then remove do16.tmp; why the touch?
Assignment 44: What is the difference between do16.tmp and do16.tmptmp?
Assignment 45: What does the sort command do, what are its -g and -k flags?
Assignment 46: What does the printf statement do? What does %s mean, and what does %7.2f mean? What's the \n?
Assignment 47: Solve assignment 41 using sort system commands.

Other useful awk commands:

```
srand()                 seed for random number generator (will take time of day)
rand()                  random number between 0 and 1
int(x)                  take the integer part of float x
man awk                 man page for awk
```

Examples:

```
      awk 'BEGIN{srand();x=3.14*rand();print x;exit}'
```

Script do17:

```
------beg
#!/bin/sh

ok=1
while [ $ok -eq 1 ]
do
  tmpdir=`awk 'BEGIN{srand();tmp="tmp";\
                     for(i=1;i<=10;i++){j=int(10*rand());tmp=tmp j;};\
                     print tmp;exit}'`
  if [ ! -d $tmpdir ]
  then
    ok=0
  fi
done
mkdir $tmpdir
------end
```

Script do17 is handy when working on local disks on clusters (these are the fastests disks, but not accessible from other nodes). You would make a temporary directory in your queueing script, copy everything to the temporary directory on the local disk, do your job, once finished, copy everything back and remove (!; don't fill up the local disk, be friendly to other users!) your tmpdir. Giving it a random name will make sure you're not writing to a directory that already exists.