ls 可以指定多个目录:ls directory1 directory2
-l 制作一个长清单，详细说明文件大小、所有权等
-C 产生多列输出
-t 按时间戳降序排列
-r Reverse the listing order
-a 列出所有文件，包括隐藏文件(文件名以. 开头)
-p 在目录的名后增加/
-S will sort files by size

ps 打印进程信息
-l 打印一个长列表包含更多信息

top 列出所有进程和 cpu 使用

kill <options> <PIDs> or pkill

gzip file_name 压缩文件，gunzip 解压

ps <options> 打印进程信息
-l 打印一个长列表包含更多信息

cut 指定分隔符分隔的多个字段
-f 指定要保留的字段
-d 后面的符号分隔列表
把 datafile.csv 按逗号分割，显示 1 和 5-7 列
```
cut -d , -f 1,5-7 datafile.csv
```

paste [-d <sep>] <files>
把两个文件的每一行分别拼成一行显示
-d 指定分隔符
合并两个文件，按照第一列倒序
```
paste -d, score.txt player.txt | sort -t',' -k lnr
```

tr <options> <string1><string2>
替换：把 string1 替换为 string2
-C 反选，不符合 string1 的才会被替换
-s 多个连续的 string1 被视为一个替换

把 textfile 的大写字母全替换为小写字母
```
tr '[A-Z]' '[a-z]' < textfile
```

#将 textfile 中连续的多个空格压缩为一个空格
```
tr -s ' ' ' ' < textfile
```

#将 windows_textfile 中的回车符转换为换行符，将 Windows 格式的文本文件转换为 Unix 格式
```
tr '\015' '\012' < windows_textfile
```

将所有小写字母转换为大写字母
```
tr 'a-z' 'A-Z' < example.txt
```

将所有空格替换为下划线
```
tr ' ' '_' < example.txt
```

comm [<output options>] <file1> <file2>
-1 不显示只在第 1 个文件里出现过的列。
-2 不显示只在第 2 个文件里出现过的列。
-3 不显示只在第 1 和第 2 个文件里出现过的列

Uniq <options> [输入文件][输出文件]：去重
-c 在每列旁边显示该行重复出现的次数。
-d 仅显示重复出现的行列。
-u 仅显示出一次的行列。
列出重复的数据(column 4)
```
tail arcade.csv -n +2 | cut -d, -f4 | sort | uniq -d
```

wc
-c # 统计字节数，或--bytes：显示 Bytes 数。
-l # 统计行数，或--lines：显示列数。
-m # 统计字符数，或--chars：显示字符数。
-w # 统计字数
-L # 打印最长行的长度

diff file1 file2: 比较文件内容差异

find 在指定目录下查找文件，默认当前目录
在当前目录下查找后缀是 txt 的,比 Alice.txt 新的，后缀是 txt 的文件
```
find -newer Alice.txt -type f -name "*.txt" -print
```

#-type f: 这个选项指定只查找文件（而不是目录、链接等）。
-name "*.txt": 这个选项指定只查找扩展名为 .txt 的文件。
-print: 这个选项指定打印出找到的文件的路径

sort [options] files 排序
-u 输出的结果是去了重的
-o <输出文件> 将排序的结果写入到文件里
-t <分割字符> 指定排序时所用的栏位分隔字符
-k 表示取某段为关键字进行排序
-n 以数字进行排序，默认升序
-r 倒序（降序）

按第四列排序，删除重复
```
sort -u -t , -k 4nr file.csv
```
-u 去重
-t, 按逗号分隔
-k 4nr 按照第四列以数字进行降序排序

打印第 999 行
```
head land.txt -n 999 | tail -n 1
```

去除表头，以逗号分隔获取第二列，排序，去重，统计行
```
tail -n+2 arcade.csv | cut -d, -f 2 | sort | uniq | wc -l
```

脚本
赋值变量<variable>=<single value>
等号两边不可以加空格
使用变量：在它前面放一个 $

$0 : the command itsel
$1: 第一个参数
$# : 参数的数量
$(( )) 直接看作十进制数的运算
$()表示将()里的内容作为命令执行
> /dev/null 丢弃输出

检查文件的第一行和最后一行是否相同。
```
#!/bin/bash
if [[ -z $1 ]] || [[ ! -e $1 ]]
then
    echo ERROR > /dev/stderr
    exit 1
fi
first=$(head -n1 $1)
last=$(tail -n1 $1)
if [[ $first == $last ]]
then
    echo "First and last lines are the same"
else
    echo "First and last lines differ"
fi
```

case
```
case $DAY in
Mon | Fri ) echo ${DAY}day ;;
Tue | Thu ) echo ${DAY}sday ;;
Wed ) echo ${DAY}nesday ;;
S??) echo WEEKEND! ;;
*) echo "$DAY is not a day I understand."
    echo "April Fool's Day?" ;;
esac
```

if [[-z "$string"]]
-n: 检查 string 是否不为空
-z: 检查 string 是否为空
-f: 检查文件是否存在并且是否为常规文件
-d: 检查路径是否存在
-eq: = -ne: != -lt: < -gt: > -le: <= -ge: >= 数字比较

for & if
#找出各文件的大小，报告其中最大的文件
```
#!/bin/bash
lsize=0
lfile=''
total=0
dir='.'
while [[ $# -gt 0 ]]
do
    case $1 in
  -h) echo -d DIRECTORY Count files in DIRECTORY instead of .
      echo
      echo -h Display this help message and exit
      exit 0;;
  -d) if [[ -z $2 ]]
      then
          echo "Missing operand for -d argument"
          exit 1
      fi
      dir=$2
      shift;;
  *) echo "Unknown argument $1"
      exit 2;;
    esac
    shift
done

for f in *; do
if [ -f "$f" ]; then
      size=$(wc -c < $f | cut -f1 -d " ")
       echo "$size $f"
       ((total+=size))
       if [[ $size -gt $lsize ]]; then
           $lsize=$size
           $lfile=$f
       fi;
    fi;
done
```

while:
#第一个参数作为文件名，第二个参数作为文件内容创建一个文本文件
```
#!/bin/bash
while [[ $# -gt 0 ]]
do
    echo $2 > $1
    shift # Shift：用于对参数的移动（左移）
    shift
done
```

grep [<options>] <regular-expression> <file>
-i 忽略大小写
-n 标示出该行的列数编号。
-v 显示不包含匹配文本的所有行。
rabbit"在 Alice_in_Wonderland.txt 中出现次数
```
grep rabbit Alice_in_Wonderland.txt | wc -l
```
or
```
grep [Rr]abbit Alice_in_Wonderland.txt | wc -l
```
Molly 的 Score 是多少
grep Molly arcade.csv | cut -d, -f6
#打印 机器为 b 且分数以数字 4 开头的所有行
grep ,b, arcade.csv | grep ,4[0-9]*$

正则表达
```
^       起始符，以^Vero 开头
$       终止符，以 nica$结尾
^$      表空行
. or ?  表匹配单个字符，且非空行
.       表 0 次或多次
.*      匹配所有内容
[ ]     从一个集合里匹配单个字符
[^ ]    除集合以外的单个字符
\       表转义。# \012 = \n
\N      表示第 N 个记录的匹配

ls /bin/g[^e]*t
#g 开头，[^e] 表示第二个字母不是 e，
 *t 表示以 t 结尾
```

sed <options> <file> …
-e 直接处理输入的文本文件
-f 以选项中指定的 script 文件来处理输入的文本文件
-n 仅显示 script 处理后的结果，关闭默认的输出
-i 直接操作文件

a： 在当前行的下一行新增行
```
sed -e 2a\hahahahah testfile
```
#在 testfile 的第二行下面新增一行

c： 取代行
```
sed -e 2c\hahahahaha testfile
```
#把 testfile 的第二行替换

d： 删除
删除 testfile 的第 2-5 行
```
sed -e 2,5d testfile
```
删除 testfile 的第 2 行和第 5 行
```
sed -e 2d -e 5d testfile
```
删除所有不包含单词 University 的行
```
sed -e '/university/I!d'
```
删除 name 字段后面包含字母的所有行
```
sed -e '/,.*[a-zA-Z].*/d'
```
将所有句号(.)替换为逗号(，)
```
sed -e 's/[.]/,/g'
```
删除所有尾随逗号
```
sed -e 's/,$//'
```

i： 在当前行的上一行新增行
#在 testfile 的第二行上面新增一行
```
sed -e 2i\hahahahah testfile
```

n： 查找
```
sed -n '/要查找的字符串/p' testfile
```

p： 打印，一般和-n 连用
```
sed -n 1,3p testfile  打印第 1-3 行
sed -n 'p;n' testfile 打印奇数行
sed -n 'n;p' testfile 打印偶数行
```
循环执行 npnpnp，就会跳过奇数行只打印偶数行
循环执行 pnpnpn，就会跳过偶数行只打印奇数行
```
sed -n 'p;n;n' testfile
```
就是打印第 1，4，7…行

l 和 p 一样都是打印，但 l 会把转义字符也打印出来

s： 替换
```
sed 's/被取代的字串/新的字串/g' testfile
```
g 表示全局替换
```
sed 's/被取代的字串/新的字串/' testfile
```
#表示只替换第一次出现的

将字段名"seat"转换为"crew"
```
sed 's/"seat"/"crew"/g' input.json
```

#替换最后一个字符不是]|{},的，将其替换为该字符后面+逗号
```
sed -e 's/\([^]|{},]\)$/\1,/' villains.json
```
s/…/…/: sed 的替换命令
regex breakdown
```
's/\(     \) /  /' # 捕获组，用于保存匹配到的内容
's/\(     \) /\1 /'      # \1 第一个捕获组的内容
's/\(     \) /\1,/'      # , 添加逗号
's/\(     \)$/\1,/'      # $ 表示文件末尾
's/\([^   ]\)$/\1,/'  # [^ ] 不想匹配的字符
's/\([^]|{},]\)$/\1,/' # 不想匹配的字符 list=][{},
```

#打印从第四行到末尾的内容
```
sed -n '4,$p' testfile
```
#. 表示当前行
 0 表示在第一行之前的行

在 JSON 文件中将键名添加双引号
```
sed -re 's/([a-z]+):/\"\1\":/' villains.json
```

#(…) 在扩展正则表达式-re 中定义了一个捕获组
Regex breakdown
```
's/…/…/'            # sed 的替换命令。
's/(   )/   /'      # 定义一个捕获组，
's/([a-z]+) /   /'  # 匹配一个或多个小写字母
's/([a-z]+):/   /'      # 匹配字段名后的冒号
's/([a-z]+):/\"\1\":/'   # \"\1\": 替换部分，将捕获的字段名加上双引号和冒号
```

sed 是用来处理行的，awk 是用来处理列的

awk 会逐行读取文本文件
$0 就是整行，$1 就是第一列，$2 就是第二列

先打印第二列，再打印第一列，用#连接
```
awk -F# '{print $2 "#" $1}' test_file
```
#-F 指定分隔符

#打印 yyyy/mm/dd
```
awk -F'[/ ]' '{print $3"/"$1"/"$2 "$4}' dates.txt
```
[/] 表示 / 或空格 作为分隔符

内置变量
```
ARGC        命令行参数个数
ARGV        命令行参数数组
FILENAME    输入文件名
FNR         当前文件的当前行索引
FS          输入的分隔符
NF          当前行有多少列
NR          多文件时，从第一个文件开始的行索引
OFS         输出的分隔符
```

格式化字符
```
%c          单字符
%s          字符串
%d          整型
%e          科学计数，默认 6 位小数
%f          浮点数，默认 6 位小数
```

#为每一行添加一个递增的行号:
printf：格式化打印（需要括号）
```
awk '{printf("%d:%s\n", ++i , $0)}' test_file
```
print：直接打印（括号可以省略）
```
awk '{print ++i ": " $0}' test_file

awk '
BEGIN { actions }
/pattern/ { actions }
/pattern/ { actions }
……..
END { actions }
' filenames
```

~ 表示匹配，!~ 表示不匹配
后接/<regular-expression>/
统计空行数，用的是正则匹配
```
awk '$0 ~ /^$/{sum++}END{print sum}'
```
统计空行数，用的是判断
```
awk '$0 == " "  {sum++}END{print sum}'
```

#打印字段数不等于 4 或者第一字段不包含 "University"
```
awk -F"," 'NF!=4 || $1 !~ /University/ {print}' universities.csv
```

#文件中的每一行，为每一行添加行号
```
awk '{print FNR "," $0 > "numbered_151.txt"}' original_151.csv
```

#打印最高生物
```
awk -F',' '{if ($2 > x) {x = $2; max = $1}} END {print max}' height.csv more_higher.csv
```

#使用 paste 和 awk 创建一个管道来连接
```
paste original_151.csv height_weight_151.csv -d"," | awk -F"," '{printf("Name: %s, Height: %.1f, Weight: %.1f, Type: %s\n", $1, $6, $7, $2)}'
```

#统计每种生物的出现次数的 types.awk 脚本
to call this script, use awk -F "," -f types.awk original_151.csv
主处理块，逐行处理输入文件
```
{
    # 增加主要类型（第 $2 字段）的计数
    types[$2]++;
    # 如果次要类型（第 $3 字段）不为空，增加其计数
    if ($3 != "")
        types[$3]++;
    }
}
# 结束块，在处理完所有行后执行
END {# 遍历 'types' 数组并打印每种类型及其计数
     for (type in types)
    print type ":" types[type]
        }
    }
```

When the script is invoked with the command:
```
echo Fred | awk -f gday.awk
```
which of the following happens:

C You see “Hullo Fred” twice, followed by “Fred”

Q2. You have been given an Awk script to fix. The script aims to sum all the numbers in a file, i.e. both across the lines of the file and also down the file. Each line can have multiple values and can vary in length.
Unfortunately, the script is not working as intended.

```
{
 for(i=1; i< NF; i++)#NF 来遍历每个字段
  sum = $i
}
END{
  printf("sum = %d\n",sum)
}
```

When you run the script on a file of integers,

a)
```
1 2 3
4 5 6
7 8 9 10
```

b)

The sum was 9 (rather than 55). What caused that problem? **[4 Marks]**

```
Ans: sum = $i should be sum += $i    (or sum = sum + $i)
```

When that error has been fixed, you rerun the code and are informed that the sum is 36. Still not right. What is the source of this problem? **[4 Marks]**

Ans: The loop test i < NF should be i <= NF (It is okay if they report the bugs in the wrong order)

Q3.
Write a Sed command that takes a file name containing spaces, and replaces each space with an underscore “_”. **[2 Marks]**

```
Ans: s/ /_/g
```

Write a Sed command that adds the string ‘.faa’ to a file name **[2 Marks]**

```
Ans: s/$/.faa/
```

Write a Sed command that adds “../data/” to the start of each file name. **[3 marks]**

```
Ans: s/^/..\/data\//
```

Q4. Write a runnable Bash script, `do_mvs` <directory> which, given two directories, looks for ordinary files (i.e. not directories) in the first directory (and recursively through its subdirectories) , and moves each of the files to the second directory. For example, assume directory `level1` contains file `a` and directory `level2`, and directory `level2` contains file `b`:

level1
    a
    level2
        b

then after `do_mvs(level1, other_directory)` the files `a` and `b` have been moved, `other_directory` now has:

a
b

while `level1` now looks like:

level 1
    level 2

The program will first need to check that the files given as the argument exist and are directories. (Hint. You may wish to consider using the `find` command.) **[12 marks]**

Ans:

```
!/usr/bin/env bash
if [[ $# -eq 0 || ! -d $1  || ! -d $2 ]]
then
  echo "usage: $0 <directory> <directory>" >
/dev/stderr
  exit 0
fi

find $1 -type f -exec mv '{}' $2 \;

Mark allocation
#!/ etc  2 marks
If statement(s) with 3 tests 4 marks
Sensible error message and exit 2
Sensible call to find 4 (don’t sweat the
punctuation)
```

Q5.
Does the string `rotor` match regular Ed-style regular expression `r..r` **[2 Marks**

It Matches It Does Not Match

Does the string `abababac` match regular Ed-style regular expression `^[ab]*$` **[2 Marks]**

It Matches It Does Not Match

Does the string `abababac` match regular Ed-style regular expression `[ab]*c[ab]*` **[2 Marks]**

It Matches It Does Not Match

Q6. Write one or more Ed style regular expressions that completely match all of the following lines. Use as few regular expressions as possible. Please note that the regular expression must involve the letters ‘a’, ‘b’, ‘c’, ‘d’ and ‘e’, but not ‘.’

```
a b c d
a a c d
a c d
a c e
```

What is/are the regular expression(s) **[4 Marks]**

```
Ans: aa*b*c[de]
```

In words, explain how do your regular expressions can be used to match all four lines. **[4 Marks]**

Ans: The first a is found in all lines, so is in the RE. a can then appear at least once more, hence a*. b does follow in one line, but mostly not, so b*. (Actually b? but that was not taught.) Then compulsory c then either d or e.

Q7. Sabine is using git to track her work. She has just updated a script called configure.sh. The old file looked like this:
configure.sh

```
#!/bin/bash

if [[ $1 -gt 9000 ]]
then
a="stardust"
else
a="duchess"
fi

device_code=$(grep "$a" codes.csv | cut -d, -f2)
./activate.sh $device_code $1
```

The updated file looks like this:
configure.sh

```
#!/bin/bash

if [[ "$1" =~ ^[1-9][0-9]*$ ]]
then
    power=$1
else
    echo "Usage: ./configure.sh <power> # power
must be a positive integer"
    exit 1
fi

if [[ $power -gt 9000 ]]
then
    device="stardust"
else
    device="duchess"
fi

device_code=$(grep "$device" codes.csv | cut -d, -
f2)
./activate.sh $device_code $power
```

a) What command (or commands) should Sabine use to record her changes in git? **[2 Marks]**

Ans: git add configure.sh (1 mark)
git commit (1 mark)
OR
git commit -a (2 marks)
OR
git add -A (1 mark)
git commit (1 mark)

b). Write a suitable git commit comment to describe Sabine's changes. **[3 Marks]**

Ans: something along the line of input validation/antibugging, improved code readability,

c). Git is a useful tool for group projects and individual projects. What is one advantage of git when used for an individual project? **[2 Marks]**

Ans: Any of
- git helps you understand past code changes
- git helps you know which version of your code is most recent
- git allows you to back up your code remotely (no need to mention github but no penalty for doing so)
- git allows you to easily revert mistakes/restore a working version

---

Q8. A computer science department noticed that there an unusual number of submissions for assignments on particular days. Luckily the department’s assignment submission system kept a log of submissions per day. You have been asked to write code to report the student IDs and the number of submissions for those unusual submissions on a particular day. The data you’ve been given is in a <tab> separated file with fields resembling:
<date> <TAB> *<time>* <TAB> *<ID>* <TAB> *<Assessment>*

You can ignore the first column as we know the data comes from a single day. Also, to keep things simple the time is just recorded as an integer, so 7:02 is 702, while 23:00 (11pm) is recorded as 2300.
For example,

```
today       900         james       Ass2
today       2302        james       Ass2
```

The input data is sorted by increasing time of the day. With that in mind, your code should look for 3 or more submissions for the assessment item
Ass2 occurring after 11pm (**23:00**), and report the corresponding IDs and number of submissions. The list of IDs should be in alphabetical order. (Don’t worry about adding antibugging code.)

Hint: You can structure this as a single Awk script, `very_suby.awk`, or an Awk script with a short Bash script `very_suby.sh` **[10 Marks]**

Ans

```
# Expected format: Date Time ID Assessment
$2 >= 2300 {if($4 == "Ass2")
                        subs_id[$3]++
}

END{
  for(id in subs_id)
    if(subs_id[id] >= 4)
            printf("%s\t%s\n", id, subs_id[id]) |
"sort -k 1n"
}
```

GIT
访问代码（将其获取到您的计算机上）
理解代码，至少在高层次上。如果幸运的话，会有一个 README 文件来解释它。
弄清楚你需要改变什么，以及你将如何改变它
以这样一种方式记录您的更改，以便它们可以传递给代码库的其他用户。
1. **配置 Git**：安装完成后，你需要配置 Git，包括设置用户名和邮箱地址。你可以使用以下命令进行配置：
   git config --global user.name "Your Name"
   git config --global user.email "youremail@example.com"

3. **初始化仓库**：要开始使用 Git 进行版本控制，你需要将你的项目文件夹初始化为一个 Git 仓库。在项目文件夹中使用以下命令：
   git init

4. **添加文件**：将你的文件添加到 Git 仓库中，以便 Git 跟踪它们的变化。使用以下命令：
   git add filename
   或者，要添加所有文件，可以使用：
git add .

5. **提交更改**：一旦你添加了文件，你就可以将它们提交到 Git 仓库，并记录这次提交的描述。使用以下命令：
   git commit -m "Your commit message"

6. **查看状态和提交历史**：你可以使用以下命令来查看 Git 仓库的状态和提交历史：
   git status
   git log

7. **创建和切换分支**：Git 允许你创建和管理分支，以便并行开发和尝试新功能。使用以下命令：
   git branch branchname # 创建分支
git checkout branchname # 切换分支

8. **合并分支**：完成一个功能后，你可以将一个分支的更改合并到另一个分支。使用以下命令：
git merge branchname

9. **远程操作**：Git 还提供了与远程仓库交互的功能，例如克隆远程仓库、拉取更新、推送本地更改等。使用以下命令：
git clone repository_url # 克隆远程仓库
git pull origin master # 从远程仓库拉取更新
git push origin master # 将本地更改推送到远程仓库

Given the file name pattern *a?*t* which of the following file names match that pattern (2 marks each):

atga	Match	Does Not Match
gctaat	Match	Does Not Match
gatt	Match	Does Not Match
atgata	Match	Does Not Match

Q2. Which of these statements is False:
- Free-to-use software is open-source
- Open-Source software is free to use
- Open-Source software is free to modify
- You are free to incorporate open-source software in other code (with due acknowledgement) (2 marks)

Q3. You are using your computer when you notice that is working hard on something; the fan is running all the time. A bit later, responses to typing have slowed markedly. In short, it is likely that a process has run off the rails.

1.Which Linux command can you use to identify the rogue process? 可以使用哪些 Linux 命令来识别非法进程?
top, htop, ps (ps not as good but info can be found there, so accept) 2 Marks

2.Which Linux command can you use to stop the process that is causing the problem, but not other processes
kill, pkill 2marks

Q4. Assume that the file a.sh contains the Shell script:

```
!/usr/bin/env bash
size=$(ls -s $1 | cut -d' ' -f 1)
if test $size -ge 10000
then
    echo $1
fi
```

Also assume that a.sh is called in the following way:

```
find ~ -name '*.gz' -exec a.sh '{}' \;
```

It may be helpful for you to know that the Unix command ls -s on some file reports the size of the file (in 512 byte blocks), followed by the file name.

What does the call to find, including the call to a.sh do (in overview, not line by line)? (4 marks)

```
Ans: Starting from the user’s home directory (home
is sufficient) (1 mark), find all files whose
names end in gz (ie gzip files) (1 mark)  and
report those which are at least 10,000 blocks in
size (2 marks)
```

Q5. What is antibugging and why is it important for programs to have it. (4 marks)

```
Antibugging is the addition of tests, typically
near the start of a program, which ensure that
that data coming from the user is consistent with
what is expected. It is important because
otherwise, nonsense results may be computed from
absent, out of range or otherwise problematic
data.
```

Q6. Write a complete, runnable Bash program that, given a text file, will print to standard output every second line, starting with the first line (then the third line etc). Please only use Bash commands. (Hint: you do not need arithmetic for this; just use a variable that is given, in rotation, one of two values. However, if you wish to use Bash arithmetic, that is also fine.) Make sure to handle errors appropriately (8 marks)

```
#!/usr/bin/env bash

if [[ ! -s $1 ]]
then
  echo "The file $1 does not exist" > /dev/stderr
# redirection not important
  exit 1  #exit important, status value is not
important
fi

odd=True
IFS="    # Don’t worry if this is not present
"
for line in $(< $1)  # while .. read also fine
do
  if [[ $odd = True ]]
  then
    echo $line
    odd=False
  else
    odd=True
  fi
done
```

Q7. Write a complete, runnable Bash program called col_count which, given the name of a single-tab-separated plain-text datafile, and a column number (from 1) – in that order – returns the item from the selected column that has the highest number of occurrences. (If there are several, any one of those with largest counts is fine.) Make sure to handle errors appropriately, but in particular include a test that the requested column number is within the range column numbers available in the file. You can assume that the first argument, if present, will be a string, and the second argument, if present, will be an integer.

For example, if datafile contains:

a	a	a
b	a	b
c	b	b
a	c	b

and the query is: col_count datafile 3
the program should report: 3 b, or just b

The framework has to be a Bash script, but it can call other Unix tools. (10 marks)

```
#!/usr/bin/env bash

if [[ $# -ne 2 ]]
then
  echo "Expecting 2 arguments <tsv text file> <col
number>" > /dev/stderr
  exit 1
fi

if [[ ! -s $1 ]]
then
  echo "$0: the nominated file $1 does not exist
or has zero length" > /dev/stderr
  exit 1
fi

col_count=$(head -1 $1 | tr '    ' '\012' | wc -l)

if test $2 -gt $col_count -o $2 -lt 1
then
  echo "Column number is greater than the number
of columns or less than 1" > /dev/stderr
  exit 1
fi

cut -d '        ' -f $2 < $1 | sort | uniq -c |
sort -k 1nr | head -1
```