cut

name,age,city

Alice,30,New York

Bob,25,Los Angeles

Charlie,35,Chicago

**cut -d ',' -f 2-3 data.txt**

age,city

30,New York

25,Los Angeles

35,Chicago

**cut -d ',' -f 1,2 data.txt**

name,age

Alice,30

Bob,25

Charlie,35

拼接两个文件

paste -d ',' file1.txt file2.txt

a,1

b,2

c,3

tr

**tr '[A-Z]' '[a-z]' < textfile**：将 textfile 中的所有大写字母转换为小写字母。

**tr -s ' ' ' ' < textfile**：将 textfile 中连续的多个空格压缩为一个空格。

**tr '\015' '\012' < windows\_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。

**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

uniq

-d：仅显示重复的行。

-u：仅显示不重复的行。

-c：统计行出现的次数。

-i：忽略大小写。

sort

-n：按数值排序。

-k：按特定字段排序。

-r：逆序排序。

-u：删除重复行。

-f：忽略大小写排序。

压缩和解压：gzip和gunzip

| **正则表达式** | **描述** | **示例** | **匹配结果** |
| --- | --- | --- | --- |

|  |  |  |  |
| --- | --- | --- | --- |
| . | 匹配任意单个字符（除换行符） | a.c | abc, a1c |

|  |  |  |  |
| --- | --- | --- | --- |
| ^ | 匹配字符串的开始 | ^abc | abc（在行首） |

|  |  |  |  |
| --- | --- | --- | --- |
| $ | 匹配字符串的结束 | abc$ | abc（在行尾） |

|  |  |  |  |
| --- | --- | --- | --- |
| \* | 匹配前一个字符零次或多次 | a\* | ``, a, aa |

|  |  |  |  |
| --- | --- | --- | --- |
| + | 匹配前一个字符一次或多次 | a+ | a, aa |

|  |  |  |  |
| --- | --- | --- | --- |
| ? | 匹配前一个字符零次或一次 | a? | ``, a |

|  |  |  |  |
| --- | --- | --- | --- |
| ` | ` | 逻辑或 | `a |

|  |  |  |  |
| --- | --- | --- | --- |
| \ | 转义字符 | \. | . |

|  |  |  |  |
| --- | --- | --- | --- |
| [abc] | 匹配 a、b 或 c 中的任意一个字符 | [abc] | a, b, c |

|  |  |  |  |
| --- | --- | --- | --- |
| [^abc] | 匹配除 a、b、c 之外的任意一个字符 | [^abc] | d, e |

|  |  |  |  |
| --- | --- | --- | --- |
| [a-z] | 匹配从 a 到 z 的任意一个字符 | [a-z] | a, b, ..., z |

|  |  |  |  |
| --- | --- | --- | --- |
| [0-9] | 匹配从 0 到 9 的任意一个数字 | [0-9] | 0, 1, ..., 9 |

|  |  |  |  |
| --- | --- | --- | --- |
| \d | 匹配任何数字 | \d | 0, 1, ..., 9 |

|  |  |  |  |
| --- | --- | --- | --- |
| \D | 匹配任何非数字字符 | \D | a, b, c |

|  |  |  |  |
| --- | --- | --- | --- |
| \w | 匹配任何字母、数字或下划线字符 | \w | a, 1, \_ |

|  |  |  |  |
| --- | --- | --- | --- |
| \W | 匹配任何非字母、数字或下划线字符 | \W | !, @, # |

|  |  |  |  |
| --- | --- | --- | --- |
| \s | 匹配任何空白字符 | \s | , \t |

|  |  |  |  |
| --- | --- | --- | --- |
| \S | 匹配任何非空白字符 | \S | a, 1, ! |

|  |  |  |  |
| --- | --- | --- | --- |
| {n} | 匹配前一个字符恰好 n 次 | a{3} | aaa |

|  |  |  |  |
| --- | --- | --- | --- |
| {n,} | 匹配前一个字符至少 n 次 | a{3,} | aaa, aaaa |

|  |  |  |  |
| --- | --- | --- | --- |
| {n,m} | 匹配前一个字符至少 n 次，至多 m 次 | a{3,5} | aaa, aaaa, aaaaa |

tail sort wc cut

**Player,Team,Machine,Password,salt,Score**

**Leon,BLUE,b,ff2889b3,related,5**

(1)有多少个队伍？（文件名arcade.csv）

tail -n +2 arcade.csv | cut -d, -f2 | sort | uniq | wc -l

(2)有多少唯一的得分值？

tail -n +2 arcade.csv | cut -d, -f6 | sort | uniq | wc -l

(3)列出所有重复的密码

tail -n +2 arcade.csv | cut -d, -f4 | sort | uniq -d

(4)哪位玩家的得分最高？

tail -n +2 arcade.csv | sort -k6,6 -t, -n | tail -n1

(5)哪个机器被最多玩家使用

tail -n +2 arcade.csv | cut -d, -f3 | sort | uniq -c | sort -n | tail -n1

#### (6)编写脚本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

ls /bin/g[^e]\*t

g[^e]\*t：g 开头，[^e] 表示第二个字母不是 e，\*t 表示以 t 结尾

(7)编写脚本 sizes.sh，打印当前目录中每个文件的统计信息：脚本应打印每个文件的文件名和文件大小。打印所有文件的数据后，打印最大文件的大小和名称，以及所有文件大小的总和

#!/bin/bash

sum=0

max\_size=0

max\_name=''

for f in \*

do

if [[ -f $f ]]

then

size=$(wc -c $f | cut -f1 -d " ")

echo "$size $f"

if [[ $size -gt $max\_size ]]

then

max\_size=$size

max\_name=$f

fi

sum=$((sum + size))

fi

done

echo "Largest is $max\_size in $max\_name"

echo "Sum is $sum"

 for f in \*：遍历当前目录中的所有文件。

 if [[ -f $f ]]：检查当前项是否为文件。

 size=$(wc -c $f | cut -f1 -d " ")：使用 wc -c 获取文件大小，并用 cut 提取文件大小的数字部分。

 sum=$((sum + size))：累计文件大小。

arithmetic

 第一行打印一个 \*，第二行打印两个 \*，依此类推。

 行数由命令行参数 -r 指定。如果没有提供参数，则打印4行。

#!/bin/bash

if [[ $# -gt 1 && $1 -eq '-r' ]]

then

    size=$2

else

    size=4

fi

row=1

while [[ $row -le $size ]]

do

    col=1

    while [[ $col -le $row ]]

    do

        echo -n '\*'

        col=$((col+1))

    done

    echo

    row=$((row+1))

done

### 修改你的三角形脚本，按照以下规则打印字符**需求：**

### 打印最后一列时，打印 \

打印第一列时，打印 |

如果行号为奇数，打印 L

否则打印 \*

#!/bin/bash

if [[ $# -gt 1 && $1 -eq '-r' ]]

then

    size=$2

else

    size=4

fi

row=1

while [[ $row -le $size ]]

do

    col=1

    while [[ $col -le $row ]]

    do

        if [[ $col -eq $row ]]

        then

            echo -n '\'

        elif [[ $col -eq 1 ]]

        then

            echo -n '|'

        elif [[ $((row%2)) -eq 0 ]]

        then

            echo -n 'L'

        else

            echo -n '\*'

        fi

        col=$((col+1))

    done

    echo

    row=$((row+1))

done

find

(1)查找并删除扩展名为 .tmp 的所有文件

find /path/to/search -type f -name "\*.tmp" -exec rm -f {} +

解释：-type f 查找普通文件，-name "\*.tmp" 匹配扩展名为 .tmp 的文件，-exec rm -f {} 删除找到的文件

(2)查找大于 100MB 的所有文件并压缩

find /path/to/search -type f -size +100M -exec gzip {} +

find /path/to/search -type f -size +100M -exec gzip {} +

(3) 查找所有 .sh 文件并修改为可执行权限

find /path/to/search -type f -name "\*.sh" -exec chmod +x {} +

解释：-name "\*.sh" 匹配所有 .sh 文件，-exec chmod +x {} 修改文件权限为可执行

(4) 查找所有 .log 文件并移动到 /backup/logs 目录

find /path/to/search -type f -name "\*.log" -exec mv {} /backup/logs/ \;

解释：-exec mv {} /backup/logs/ 将找到的文件移动到指定目录

(5)查找并删除所有空目录

find /path/to/search -type d -empty -exec rmdir {} +

解释：-type d 查找目录，-empty 匹配空目录，-exec rmdir {} 删除空目录。

(6) 查找最近 7 天内修改的所有文件

find /path/to/search -type f -mtime -7

(7) 查找所有属于用户 username 的文件

find /path/to/search -type f -name "\*.conf" -exec cp {} /backup/conf/ \;

解释：-exec cp {} /backup/conf/ 将找到的文件复制到指定目录

(8) 查找大小在 1MB 到 10MB 之间的文件

find /path/to/search -type f -size +1M -size -10M

解释：-size +1M 查找大于 1MB 的文件，-size -10M 查找小于 10MB 的文件

(9) 查找所有 .txt 文件并执行自定义脚本 process.sh

find /path/to/search -type f -name "\*.txt" -exec ./process.sh {} +

解释：-exec ./process.sh {} 对找到的文件执行自定义脚本 process.sh。

grep

(1)rabbit”在Alice\_in\_Wonderland.txt中出现次数

grep rabbit Alice\_in\_Wonderland.txt | wc -l #5

grep Rabbit Alice\_in\_Wonderland.txt | wc -l#47

(2)同时搜索

grep -i rabbit Alice\_in\_Wonderland.txt | wc -l # note this also matches RABBIT

#or

grep [Rr]abbit Alice\_in\_Wonderland.txt | wc -l

(3)有多少行不包含“Caterpillar”或“caterpillar”一词？

grep -vic caterpillar Alice\_in\_Wonderland.txt

(4)Alice\_in\_Wonderland.txt 中“Alice”一词出现了多少次？请注意，它有时会在同一行上出现多次。您可能想查看 grep -o

grep -o Alice Alice\_in\_Wonderland.txt | wc -l # 401

**下面用grep命令解决arcade.csv的问题**

(5)打印 arcade.csv 中团队为绿色的所有行

grep GREEN arcade.csv

(6)Molly的Score是多少

grep Molly arcade.csv | cut -d, -f6

(7)打印 arcade.csv 中机器为 b 且分数以数字 4 开头的所有行

grep ,b, arcade.csv | grep ,4[0-9]\*$

sed

**您已收到一份大学入学数据列表 (ustralian-universities.csv)，但数据很混乱。编写一个 sed 脚本来执行以下操作**

(1)删除所有不包含单词 University 的行（不区分大小写，因此您还应该保留 UNIversity 等）

/university/I!d

(2)删除名称字段后包含字母的所有行

/,.\*[a-zA-Z].\*/d

(3)将所有句号 (.) 替换为逗号 (,)

s/[.]/,/g

(4)删除所有尾随逗号

s/,$//

===华丽的分割线===

(5)将文件 file.txt 中的第一个 foo 替换为 bar

sed 's/foo/bar/' file.txt

(6)将文件 file.txt 中的所有 foo 替换为 bar

sed 's/foo/bar/g' file.txt

(7)只将 file.txt 的第二行中的第一个 foo 替换为 bar

sed '2s/foo/bar/' file.txt

(8)将替换后的内容输出到 output.txt 文件中。

sed 's/foo/bar/' file.txt > output.txt

(9)在 file.txt 每一行的末尾添加 end

sed 's/$/ end/' file.txt

(10)在 file.txt 每一行的开头添加 start

sed 's/^/start /' file.txt

(11)删除 file.txt 的第三行。

sed '3d' file.txt

(12)删除 file.txt 中包含 pattern 的行

sed '/pattern/d' file.txt

(13)仅打印 file.txt 的第五行

sed -n '5p' file.txt

(14)仅打印 file.txt 中包含 pattern 的行

sed -n '/pattern/p' file.txt

(15)将 file.txt 中的所有 foo 替换为 bar，忽略大小写

sed 's/foo/bar/I' file.txt

(16)在一次命令中使用多个 sed 替换命令

sed -e 's/foo/bar/g' -e 's/baz/qux/g' file.txt

(17)在 file.txt 的第二行后插入 This is a new line

sed '2a\This is a new line' file.txt

(18)在 file.txt 的第二行前插入 This is a new line

sed '2i\This is a new line' file.txt

(19)将 file.txt 中的 foobar 替换为 barfoo，使用捕获组

sed 's/\(foo\)\(bar\)/\2\1/' file.txt

comm

**file1.txt**

apple

banana

cherry

date

**file2.txt**

banana

cherry

fig

grape

(1)显示三个列

comm file1.txt file2.txt

输出

apple

date

banana

cherry

fig

grape

(2) 只显示第一个文件的独特行

comm -23 file1.txt file2.txt

输出

apple

date

(3) 只显示两个文件共有的行

comm -12 file1.txt file2.txt

输出

banana

cherry

(4) 只显示第二个文件的独特行

comm -13 file1.txt file2.txt

输出

fig

grape

(5)只显示不同的行

comm -3 file1.txt file2.txt

输出

apple

date

fig

grape

awk

(1)有mm/dd/yyyy 格式的生日列表，现在用awk打印日期为dd/mm/yyyy 格式

awk -F / '{print $2"/"$1"/"$3}' american\_dates.txt

**Monash University,42339,22140,64479**

(2)得到第三行到第六行的数据

awk 'NR==3,NR==6{print $0}' australian-universities.csv

(3)得到第三列和第二列的数据

awk '{print "third line: "$3,"second line: "$2}' australian-universities.csv

输入分隔符，默认为空格，空白字符，英文是field separator，变量名是FS

输出分隔符，output field separator，简称OFS

**Entity Code Year GDP per capita, PPP (constant 2017 international $) Population (historical estimatesHomicide rate per 100,000 population - Both sexes - All ages Life expectancy - Sex: all - Age: at birth - Variant: estimates Cantril ladder score**

**Afghanistan AFG 2011 1961.0963 29249156 4.208668 61.4 4.25835**

(4)第二列和第三列

awk -F "\t" -v OFS="=====" '{print $2,$3}' sample2.tsv

(5)第四行到第八行

awk -F "\t" -v OFS="=====" 'NR==4,NR==8{print $2,$3}' sample2.tsv



(6)输出第二列，把行号和列号也输出出来

awk -F "\t" -v OFS="====" '{print NR,$2}' sample2.tsv

(7)输出以university开头的行

awk -F "," '/^university/{print $0}' australian-universities.csv

(8)打印文件 data.txt 中包含单词 "apple" 的行

awk '/apple/' data.txt

(9)打印文件 data.txt中的每一行及其行号

awk '{print NR, $0}' data.txt

(10)统计文件words.txt中每个单词的出现次数

awk '{count[$1]++} END {for (word in count) print word, count[word]}' words.txt

(11)打印文件data.txt中第二个字段大于100的行

awk '$2 > 100' data.txt

(12)将所有第二个字段包含 "error" 的行的第三个字段替换为 "fixed"

awk '$2 ~ /error/ {$3 = "fixed"} {print}' data.txt

(13)合并第二和第三个字段为一个新字段，并按这个新字段的字母顺序排序后输出

awk '{new\_field = $2 $3; print new\_field, $0}' data.txt | sort -k1,1 | awk '{print $2, $3, $4}'

(14)计算文件中每个唯一类别（第一列）的第二个字段的平均值

awk '{sum[$1] += $2; count[$1]++} END {for (i in sum) print i, sum[i]/count[i]}' data.txt

Q1.The Awk script gday.awk contains the following [2 marks]

BEGIN{

array[0] = "Hullo"

array[1] = "Hullo"

}

{for(i=0; i<=2; i++)

print array[i] " " $1

}

When the script is invoked with the command:

echo Fred | awk -f gday.awk which of the following happens:

A You see “Hullo Fred” twice

B You see “Hullo Fred” three times

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

D There is a runtime error because a variable has been referenced without having previously been assigned a value.

Ans:C

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++)

sum = $i

}

END{

printf("sum = %d\n",sum)

}

When you run the script on a file of integers,

1 2 3

4 5 6

7 8 9 10

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

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

1. 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.

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

Ans: s/ /\_/g

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

Ans: s/$/.faa/

1. 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 \;

 find "$1" -type f：递归查找第一个目录中的所有普通文件。

 -exec mv '{}' "$2" \;：对于找到的每个文件，执行 mv 命令将其移动到第二个目录。{} 是占位符，代表当前找到的文件，\; 结束 -exec 命令。

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 ababababc match regular Ed-style regular expression ^[ab]\*$ **[2 Marks]**

It Matches It Does Not Match

Does the string ababababc 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

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

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

1. 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 anindividual 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"

}

Another (more likely) solution is to create an Awk script, but without the internal call sort, and place the call to the awk script in a Bash script, with the call to Awk followed by a call to sort.

Marks breakdown:

4 marks for gathering subs which are post 23:00 and for Ass2

3 for looping through to get those with at least 4 hits

3 for the final sort



A screenshot of a computer

Description automatically generated

**Optional Test**

Q1. 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.

Which Linux command can you use to identify the rogue(流氓) process?

*top, htop, ps (ps not as good but info can be found there, so accept) 2 Marks*

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)**

*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