AWK, GRP, SED tutorial

This tutorial covers only the basics of AWK, GRP, SED tools
and regular expressions commonly used in Unix shell scripting.

Introduction to AWK

An **awk** program is a sequence of patterns and actions that tell what
to look for in the input data and what to do when it is found.

Display user names from /etc/passwd (field 1):

```
awk -F: '{ print $1 }' /etc/passwd
```

Where F is the field separator; in passwd file, fields are separated by ':'
Default field separator is a blank space. Awk scans the input file and
splits each input line into fields.

Similarly:

```
cat /etc/passwd | awk -F: '{ print $1 }'
```

Display user names home directories and login shell (fields 1 and 7):
and store them in a separate file, users.txt

```
awk -F: '{ print $1, $6, $7 }' /etc/passwd > users.txt
```

or

```
cat /etc/passwd | awk -F: '{ print $1, $6, $7 }' > users.txt
```

Default field separator is empty space. To print users (field 1) from just
created file users.txt:

```
awk '{ print $1 }' users.txt
```

Introduction to GREP

grep is used to search files or standard input for lines containing
required patterns.
We'll work with a text file, **list.txt**, containing the following text:

```
Check the inode list today
reboot the machine tomorrow
  Reboot it again in a week
Call Tech support in case of emergency.
tel: 834

Oop 0
Oops 1
Oopss 12
Oopsss 123
Oopssss  1234
End
```

To get the line containing string "inode" in file list.txt:

```
grep inode  list.txt
```

To get the line containing "inode lis " in file list.txt:

```
grep "inode lis " list.txt
```

It should give you nothing as there is no string " lis "

To search for the line containing "inode list" in all the files in current directory:

```
grep "inode list" *
```

Syntax of grep: grep [options] regex [files]
where regex are regular expressions.

Using regular expressions

Regular expressions: Literals (plain text or literal text),
 metacharacters (special meaning characters).

When you construct regular expressions, you use metacharacters and literals
to specify three basic ideas about your input text:
position anchors, groups, ranges and quantity modifiers.

```
Anchors: ^  -match at the beginning of a line
         $  -match at the end of a line
```

```
grep '^Call' list.txt
grep '^  Reboot' list.txt
grep 'today$' list.txt

Count the number of empty lines:
grep -c '^$' list.txt

Display all lines containing only the word End by itself:
grep '^End$' list.txt
```

```
Groups and ranges: [abc]   -match any single character from a,b or c
                   [a-e]   -match any single charcter from among the range a-e
                   [^abc]  -inverse match, matches a single character not
                            among a,b, or c.
                   [^a-e]  -inverse match, matches a single character not from
                            the range a-e
               \< word\>   -match word
             . (single dot) -match any single character among a new line
                   \        -turn off the special meaning of the character
                            that follows
```

```
grep '[Rr]eboot' list.txt
grep '\<[Rr]eboot\>' list.txt

Display all lines from file list.txt which contain thre adjucent digits:
grep '[0-9][0-9][0-9]' list.txt

Display the lines with four or more characters in the line:
grep '....' list.txt

Display all non-blank lines from file list.txt:
grep '.' list.txt

Display all lines that contain a period:
grep '\.' list.txt
```

```
Modifiers:  *  -match zero or more instance of the preceding single character
            ?  -match zero or one  instance of the preceding regex
```

```
                (implies 'grep -E' option).
          +   -match one or more  instance of the preceding regex
                (implies 'grep -E' option).
       \{n,m\} -match a range of occurrences of the single character or regex
                 that precedes this construct; \{n\} matches n occurences;
                 \{n,\} matches at least n occurences.
          |   -match either the regex specified before or after the
                vertical bar (implies 'grep -E' option).
```

Display all lines from list.txt that contain Oop, Oops, Oopss, and so on:

```
grep 'Oops*' list.txt
```

Display all lines from list.txt that contain Oops, Oopss, and so on:

```
grep 'Oopss*' list.txt
```

Display all lines from list.txt that contain two or more adjacent digits:

```
grep '[0-9][0-9][0-9]*' list.txt
```

Display all lines from list.txt that contain '3' or '34' number combination:

```
grep -E '34?' list.txt
```

Display all lines from list.txt containing at least one digit:

```
grep -E '[0-9]+' list.txt
```

Display all lines from list.txt containing sss and ssss:

```
grep 's\{3,4\}' list.txt
```

Display all lines from list.txt containing any three, four or five digit numbers:

```
grep '\<[0-9]\{3,5\}\>' list.txt
```

Display all lines from list.txt containing "Reboot", "reboot" or "support" strings:

```
grep -E '[Rr]eboot|support' list.txt
```

Display all lines from list.txt containing any letter (no empty lines):

```
grep '[A-Za-z]' list.txt
```

Display all lines from list.txt containing any non alpha-numeric and space symbol:

```
grep '[^ 0-9A-Za-z]' list.txt
```

Display all lines from list.txt containing uppercase letter, followed by zero or more lowercase letters:

```
grep '[A-Z][a-z]' list.txt
```

Display all lines from list.txt containing 3 digit telephone number:

```
grep 'tel: [0-9]\{3\}' list.txt
```

Introduction to SED

String editor, **sed**, is used for editing lines in a file or a stream; output is going to the standard output and can be re-directed to a new file.

Syntax:
```
sed [options] 'command1' [files]
sed [options] -e 'command1' [-e command2 ...] [files]
sed [options] -f script [files]
```

Delete lines from 3 through 5 in file list.txt:

```
sed '3,5d' list.txt
```

Delete lines that contain "0" at the beginning of the line:

```
sed '/^0/d' list.txt
```

Translate capital C,R,O into small c,r,o:

```
sed 'y/CRO/cro/' list.txt
```

Delete ampty lines:

```
sed '/^$/d' list.txt
```

Replace string Oop with Wee for the first occurence on a line

```
sed 's/Oop/Wee/' lsst.txt
```

Remove ss string (replace with empty entry)for the first occurence on a line:

```
sed 's/ss//' list.txt
```

Remove ss string for all occurences on a line:

```
sed 's/ss//g' list.txt
```

Substitute a single space for any number of spaces wherever they occur on the line:

```
sed 's/ */ /g' list.txt
```

Substitute underscore for any number of spaces wherever they occur on the line:

```
sed 's/ */_/g' list.txt
```

Questions: explain what the following commands are doing and find a way to improve them and rewrite them accordingly.

a)

```
for mcfile in P4f_=ww_=h_=LR P4f_=zz_=h_=LR P4f_=zzorww_=h_=LR P4f_=Higgs_=h_=LR
do
    echo $mcfile > tmp1.txt
    cat tmp1.txt | cut -f1-3 -d"_" > tmp2.txt
    cat tmp2.txt | sed 's/=//g' > tmp3.txt
    cat tmp3.txt | sed 's/$/_PR/'
done
```

Rewrite it with ONE line of codes inside loop using AWK command (no SED) without any tmp*.txt files.

b)

```
#!/bin/bash
cat >> test.txt << EOF
/ is the root directory
bin is the system binary directory
usr bin is the user binary directory
home $USER is my home directory
home $USER bin is my binary directory
EOF
cat test.txt | sed 's/^bin/\/bin/' > tmp1.txt
cat tmp1.txt | sed 's/usr bin/\/usr\/bin/' > tmp2.txt
cat tmp2.txt | sed 's/home/\/home/' > tmp3.txt
cat tmp3.txt | sed 's/ '$USER'/\/'$USER'/' > tmp4.txt
cat tmp4.txt | sed 's/ bin/\/bin/'
```

Again, rewrite it without any tmp*.txt files. Also, did you see any problem after executing the original script? how to fix?

c)

```
for number in 0 1 2 3
do
cat > test${number}.txt << EOF
this is the $number txt file generated
EOF
done
for number in 0 1 2 3
do
       if [ $number -eq 0 ]; then
              cat test0.txt | sed 's/0/first/' > tmp.txt
              cat tmp.txt
              mv tmp.txt test0.txt
```

```
        elif [ $number -eq 1 ]; then
                cat test1.txt | sed 's/1/second/' > tmp.txt
                cat tmp.txt
                mv tmp.txt test1.txt
        elif [ $number -eq 2 ]; then
                cat test2.txt | sed 's/2/third/' > tmp.txt
                cat tmp.txt
                mv tmp.txt test2.txt
        elif [ $number -eq 3 ]; then
                cat test3.txt | sed 's/3/forth/' > tmp.txt
                cat tmp.txt
                mv tmp.txt test3.txt
        fi
done
rm -f testc.txt
for number in 0 1 2 3
do
    cat test${number}.txt >> testc.txt
done
echo
cat testc.txt
```

Rewrite the above codes with only ONE loop and do it without any tmp*.txt files or IF statements. ‘test*.txt’ files should be generated first and then modified. The print output to the screen should stay the same.