# 1. Basic Shell Commands

- Which command is the most useful?

To me "mv" is the most useful command because the file I download from a website usually automatically get saved in the folder "Downloads". For example, I might want to move the lecture slide downloaded to my 37820 class folder. In that case, using the command "mv" comes very handy.

- Which command has unexpected or surprising features?

The command "less" is kind of surprising because I originally thought that "more" and "less" should have opposite effect. It turns out that "more" does not allow me to scroll backwards through a file while "less" actually makes it possible. In some sense, "less" is more flexbile than "more".

# 2. Interactive Python Using the Notebook

## Interactive Python

Question 1

In [80]:
```
def a_counter(text):
        return sum([i=='a' for i in text])

# test 1
str1='I am an adder. 1 plus 2 is 3.'
a_counter(str1)
```

Out[80]: 3

In [81]:
```
# test 2
str2='Second check on adder function a_counter'
a_counter(str2)
```

Out[81]: 2

In [82]:
```
# test 3
str3="All a's to ace with capital A"
a_counter(str3)
```

Out[82]: 4

Question 2

In [83]:
```
def my_counter(text):
        counts={}
        for i in text:
                counts[i]=counts.get(i,0) + 1
        return counts

# test 1
my_counter(str1)
```

Out[83]:
```
{' ': 8,
 '.': 2,
 '1': 1,
 '2': 1,
 '3': 1,
 'I': 1,
 'a': 3,
 'd': 2,
 'e': 1,
 'i': 1,
 'l': 1,
 'm': 1,
 'n': 1,
 'p': 1,
 'r': 1,
 's': 2,
 'u': 1}
```

In [84]:
```
# test 2
my_counter(str2)
```

Out[84]:
```
{' ': 5,
 'S': 1,
 '_': 1,
 'a': 2,
 'c': 5,
 'd': 3,
 'e': 4,
 'f': 1,
 'h': 1,
 'i': 1,
 'k': 1,
 'n': 5,
 'o': 4,
 'r': 2,
 't': 2,
 'u': 2}
```

## Simplest batch mode Python

In [85]:
```
# %load my_code.py
from math import *

a=2*pi*5
print("My name is Siying C.")
```

```
My name is Siying C.
```

## Simple text mining with Python

In [86]:
```
import urllib.request

# Read in file
url_example="http://www.stat.uchicago.edu/~meiwang/courses/lincoln.txt"
fh=urllib.request.urlopen(url_example)

# One big string: from byte to string, skip the first four lines that are title, au
thor, and empty lines
for i in range(1,5):
        text_example=fh.readline()
text_example=fh.read().decode()
print(text_example)
```

```
Fourscore and seven years ago our fathers brought forth on this continent, a new
nation, conceived in Liberty, and dedicated to the proposition that all men are c
reated equal.

Now we are engaged in a great civil war, testing whether that nation, or any nati
on so conceived and so dedicated, can long endure. We are met on a great battle-f
ield of that war. We have come to dedicate a portion of that field, as a final re
sting place for those who here gave their lives that that nation might live. It i
s altogether fitting and proper that we should do this.

But, in a larger sense, we can not dedicate - we can not consecrate - we can not
hallow - this ground. The brave men, living and dead, who struggled here, have co
nsecrated it, far above our poor power to add or detract. The world will little n
ote, nor long remember what we say here, but it can never forget what they did he
re. It is for us the living, rather, to be dedicated here to the unfinished work
which they who fought here have thus far so nobly advanced. It is rather for us t
o be here dedicated to the great task remaining before us - that from these honor
ed dead we take increased devotion to that cause for which they gave the last ful
l measure of devotion - that we here highly resolve that these dead shall not hav
e died in vain - that this nation, under God, shall have a new birth of freedom -
 and that government of the people, by the people, for the people shall not peris
h from the earth.
```

***NOTE: I strip the first four lines which are not the body of the speech***

In [87]:
```
# Function to count the occurrences of each word in a given text file
def word_counter(text):
        text_lst=text.split()
        text_lst=[a for a in text_lst if a!='-']
        counts={}
        for i in text_lst:
                counts[i]=counts.get(i,0) + 1
        return counts

lincoln_dic=word_counter(text_example)
```

Calculate the total number of words in Lincoln's speech

In [88]:
```
sum(lincoln_dic.values())
```

Out[88]: 270

***NOTE: I ignore the hyphen '-' when counting words, except when the hyphen is used as a conjunction of two words***

Find the top five most frequently used words and their counts

In [89]:
```
sorted(lincoln_dic.items(), key=lambda x:x[1], reverse=True)[:5]
```

Out[89]:
```
[('that', 13), ('the', 9), ('we', 8), ('to', 8), ('a', 7)]
```

Determine if a string is in a file

In [90]:
```
def string_in_file(text, string):
        return string in text

# test 1
string_in_file(text_example, "seven years ago")
```

Out[90]:
```
True
```

In [91]:
```
# test 2
string_in_file(text_example, "eight years ago")
```

Out[91]:
```
False
```

In [92]:
```
# test 3: check whether '-' as a conjuction is splitted out
string_in_file(text_example, "battle-field")
```

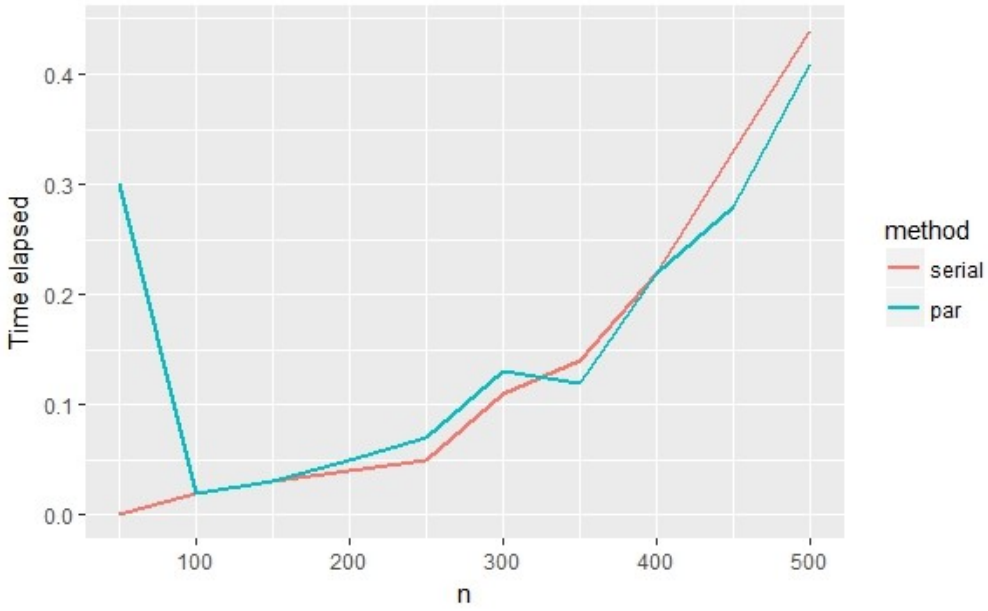
Out[92]:
```
True
```

# 3. Parallel Computing Using R

## Commented code

In [ ]:
```
# Load the packages
library(doParallel)
library(doRNG)
library(ggplot2)
library(data.table)
set.seed(111)

# Prepare for parallel
detectCores()
detectCores(logical=F)

cl <- makeCluster(3)

registerDoParallel(cl)
getDoParWorkers()

# Primitives
K <- 10

MatrixInverse <- function(n){
        M <- matrix(rnorm(n^2), ncol=n, nrow=n)
        return(solve(M))
}

CreateThreeMatrices <- function(n){
        for (i in 1:3){
                MatrixInverse(n)
        }
}

## Parallel computing
time_vec_par=numeric(length=10)
for (k in 1:K){
        start=proc.time()
        foreach(i=1:3) %dorng% MatrixInverse(50*k)
        time_vec_par[k]=proc.time()["elapsed"]-start["elapsed"]
}

stopCluster(cl)

## Serial computing
time_vec <- numeric(length = K)
for (k in 1:K){
        start=proc.time()
        CreateThreeMatrices(50*k)
        time_vec[k] <- proc.time()["elapsed"]-start["elapsed"]
}
print(time_vec)
print(time_vec_par)

dt <- data.table(n=50*(1:K), serial=time_vec, par=time_vec_par)
dt.long <- melt(dt, id.vars="n", measure.vars=c("serial","par"),
                variable.name="method", value.name="time")

g <- ggplot(data=dt.long, aes(x=n, y=time, group=method, color=method))
g + geom_line(size=1, linetype=1) +
        xlab("n") +
        ylab("Time elapsed") +
        theme_grey()
```

## Plot the computing time and compare




## Comment

When the dimension of the matrix is only moderate, there is no obvious advantage in using parallel computing. In fact, for small matrices, the extra time for triggering the parallel job, including assigning the clusters, tends to dominate the potential gain. Hence, there is no need for using parallel computing method if the task is not extremely heavy. As the matrix gets bigger, however, the gain from parallel processing offsets the initial loss, thereby saving the total computation time substantially. For any task, we should first take a stand of whether it's worth going parallel, and if the answer is yes, determine whether parallel computing is feasible for that particular task.

In [ ]: