Fundamentals in Computational Analysis of Large-Scale Datasets

3 March 2025

Thorfinn Korneliussen/Julian Perez

Korneliussen group - Section for GeoGenetics at the GLOBE Institute

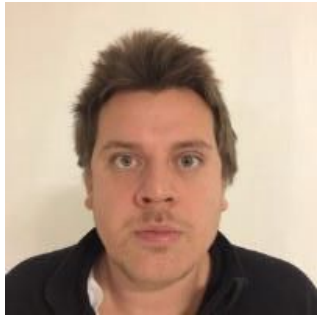
Abigail Daisy Ramsøe

Staff Bioinformatician

- BSc Computer Science
- MSc in Bioinformatics
- PhD in Archaeology

About me

- Bachelor Mathematics and computer-science. 2008
- Master Bioinformatics. 2012
- PhD at NHM. 2015
- PostDoc at Cambridge, UK. 2017
- Assistant Professor NHM. 2018
- *Section for GeoGenetics* at the GLOBE institute



I implement and develop method for the analyses of aDNA and low-coverage sequencing data.

Section for GeoGenetics at the GLOBE Institute

Julian Regalado Perez

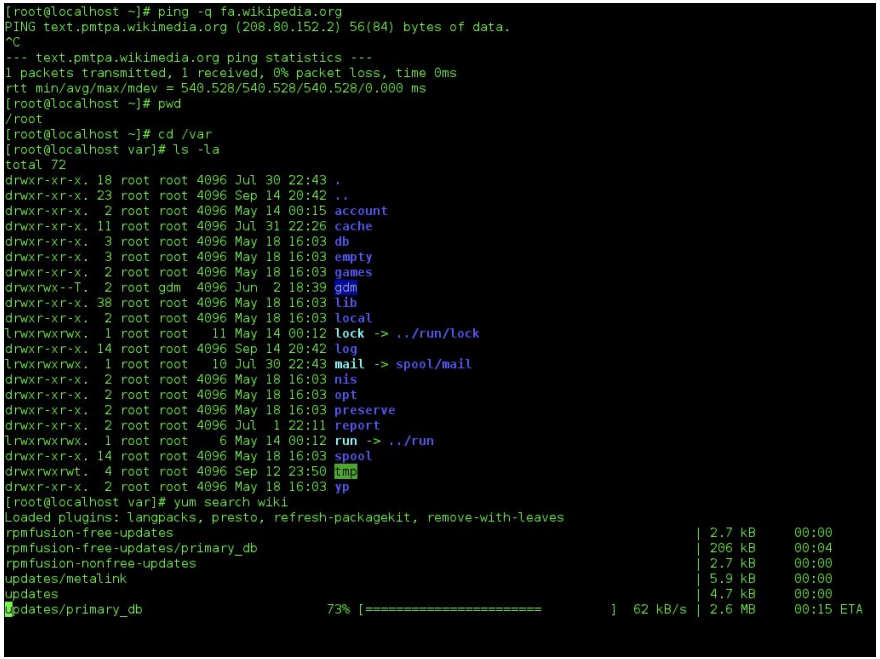
Staff Bioinformatician

- BSc Genomic Sciences, UNAM
- PhD Bioinformatics, Max Planck, Tübingen
- Postdoc- Gopalakrishnan group

I maintain compute infrastructure, implement high performance code and miscellaneous bioinfo tasks.

Overview: Day 1-UNIX shell (3 March)

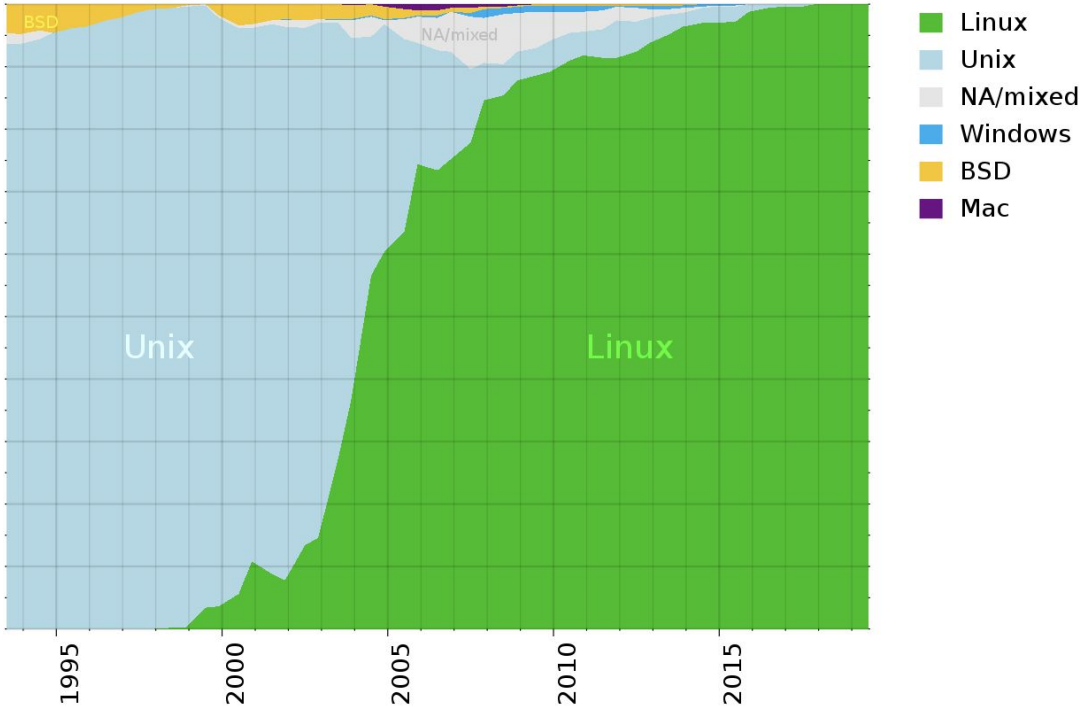
- **Back to basics**
- UNIX and POSIX
- Command line interface (CLI), *shell*
- Filesystem, Files, Directories, paths
- Programs, processes, job control
- stdin/stdout, pipes, filters
- Loops, scripts
- Transferring files between systems



Main interface for accessing servers. Running programs on servers. CLI is also called the shell.

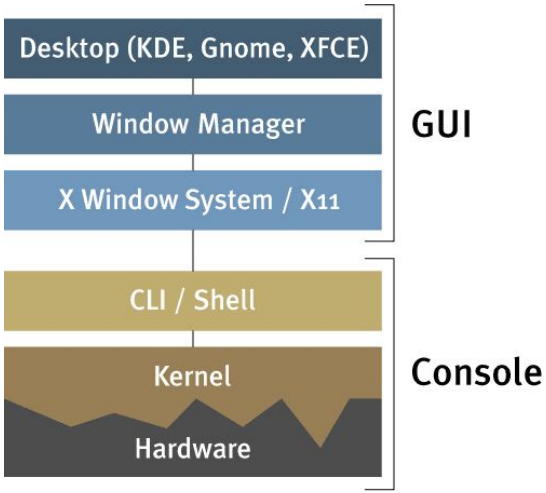
Operating systems: Windows, macOS, UNIX, and Linux

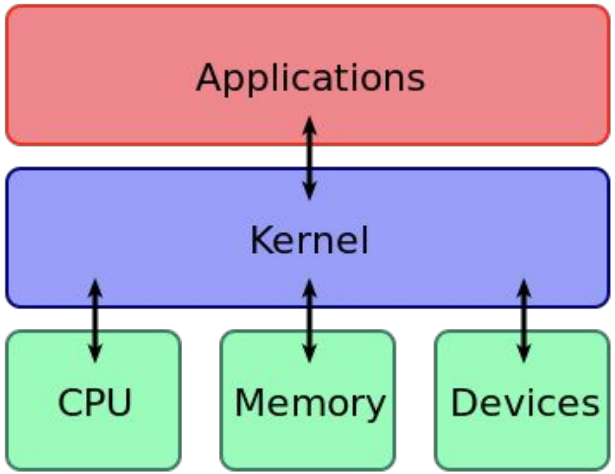
- Most desktop computers are running windows: 77%-87.8%
- All top500 supercomputers in the world is running Linux. (see Figure to the right)
- UNIX systems are everywhere: smartphones, servers, apple laptops, fridges.
- UNIX is access through a shell rather than graphical user interfaces (GUI)
- Shells are used for automating menial tasks
- Shell knowledge is essential for data analysis



Basic concepts

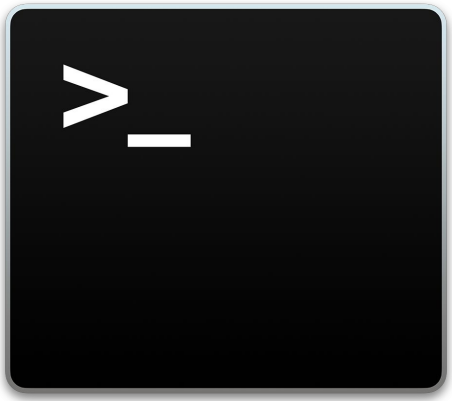
• Hardware/Software: Hardware is the physical part, software are the programs or instructions that ‘run’ on the hardware
• Focus will be programs and software that are started by ‘users’ and will run on the linux server(the kernel)





Operating system: Low level software that controls the hardware, scheduling of processes, access to network, screen, mouse etc.

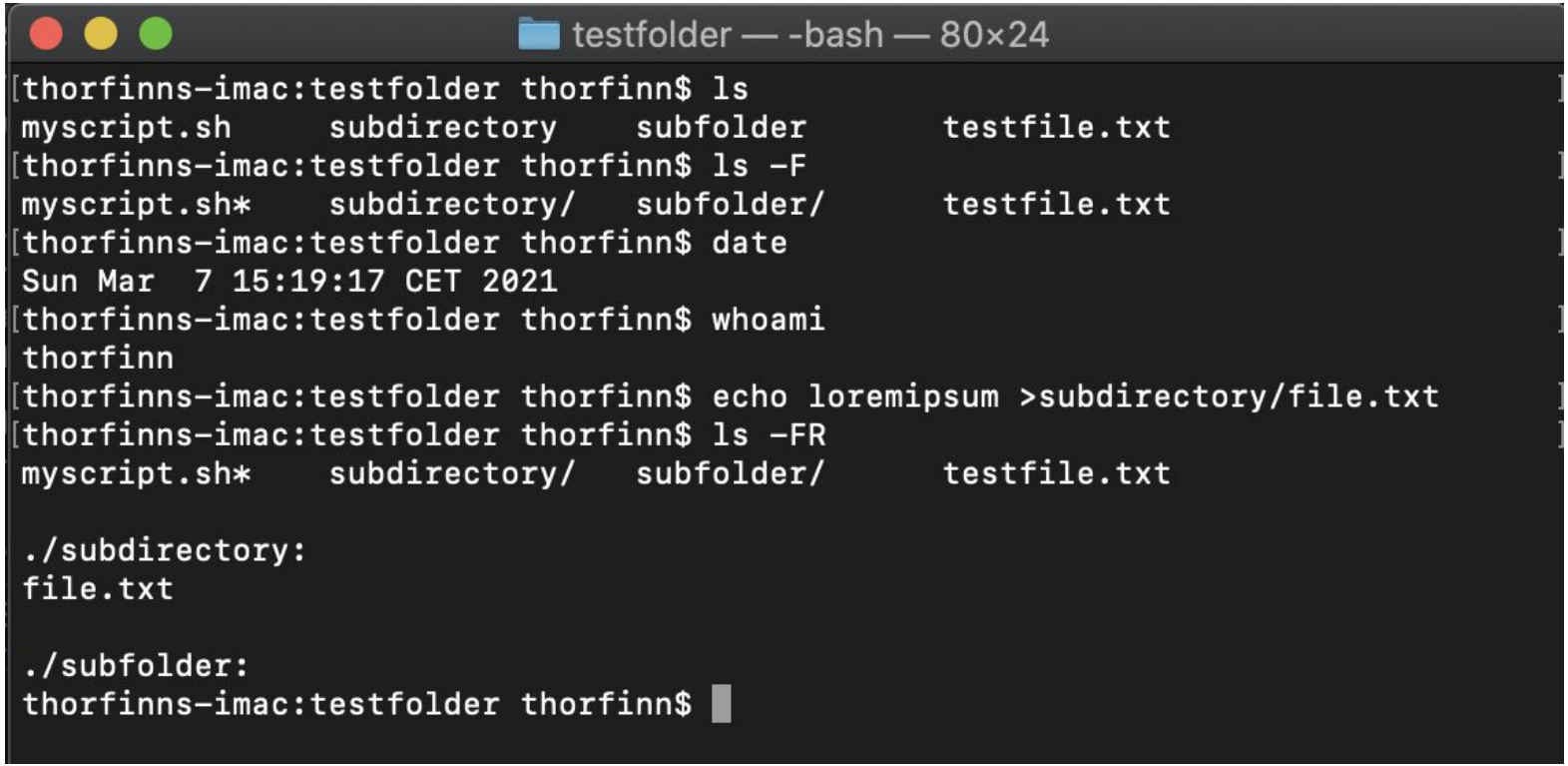
Our operating system is UNIX (follows POSIX standard) which means it has to have a shell with a number of tools supplied by the operating system.



- Begins with prompt. Server is now ready for input.
- Type command press <enter>
- The command is executed
- A new prompt is displayed ready for next input
- Sometimes the prompt starts with $ instead of >

(When a program is executed it is a process)

Example of shell commands

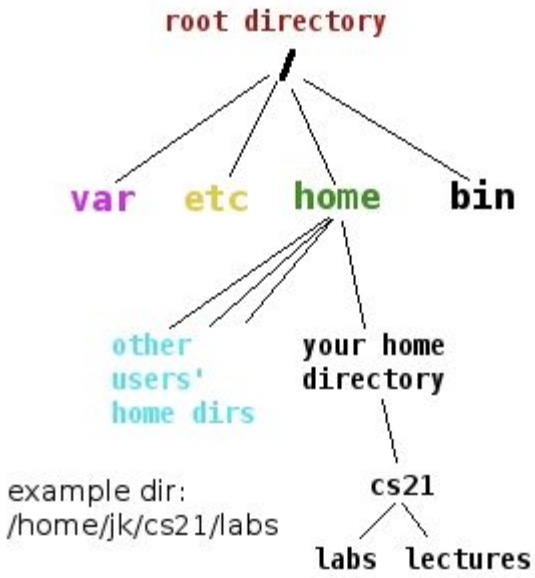


UNIX Files and filesystem 1/3

- **Computer file:** Is a sequence of bytes that resides on some storage medium.
- **Pathname:** Is a name that uniquely identifies a computerfile in a filesystem.
- **Directory**: Is a container for a group of files. A directory can can be included in other directories.
- For example **dir1/hello,** refers to the file called hello that resides in folder/directory dir1
- For example **/dir1/dir2/dir3/README.txt**. /dir1/dir2/dir3 is the dirname, basename is README.txt it has the **extension** txt.
- Please note the difference between **absolute path and relative path**.
- When you log on to a UNIX system you are put into your **home** directory **$HOME**.

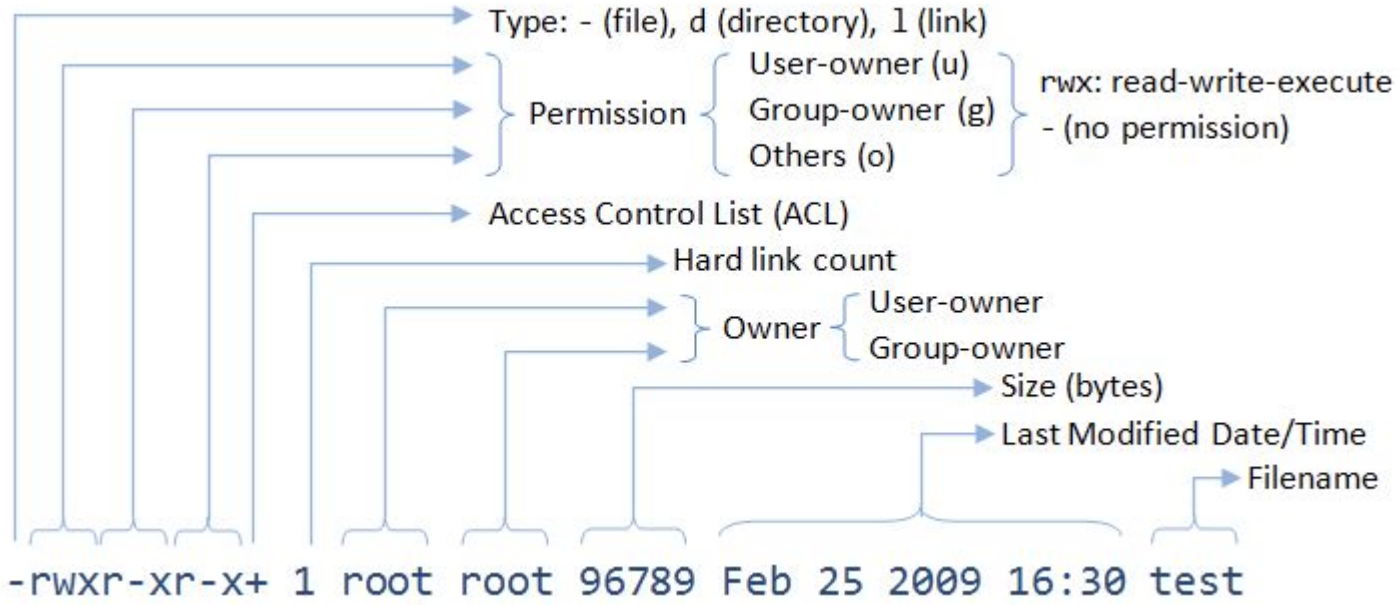
UNIX Files and filesystem 2/3

- List files and directories, **ls**
- Remove/delete a file, **rm**
- Make or create a directory **mkdir**. Delete a directory **rmdir**
- Change to directory, **cd** (change directory)
- Step one level up towards the root, **cd ..,** step two steps up, **cd ../../**
- Get current directory, **pwd**
- Copy a file, **cp** source destination
- Copy a directory (with contents, called recursive), **cp -r** sourcedir destinationdir
- Move a file, **mv** source destination
- Print contents of file **cat** or **tac** or interactively **less**
- Count the number of words, and lines **wc** (wordcount)



UNIX Files and filesystem 3/3

>ls -l test



PRE Exercise 1: Software carpentry (shell)

https://swcarpentry.github.io/shell-novice/

- Download the data
- Install the shell

Exercise 1: Software carpentry (shell)

If you have never heard about these concepts before, none of it will have made sense until you have tried it yourself.

THE SETUP:

You are ‘Nelle Nemo’, a marine biologist. Your supervisor has given you a great project: but you have to use *her* analysis tools, and they are command line tools that only work on Unix machines...

GO HERE: https://swcarpentry.github.io/shell-novice/

1. Introducing the Shell
2. Navigating Files and Directories
3. Working With Files and Directories

Boxes in orange will indicate questions that you should not run but try to resolve by thinking

Blue boxes have commands you should try to run. The output is shown in grey.

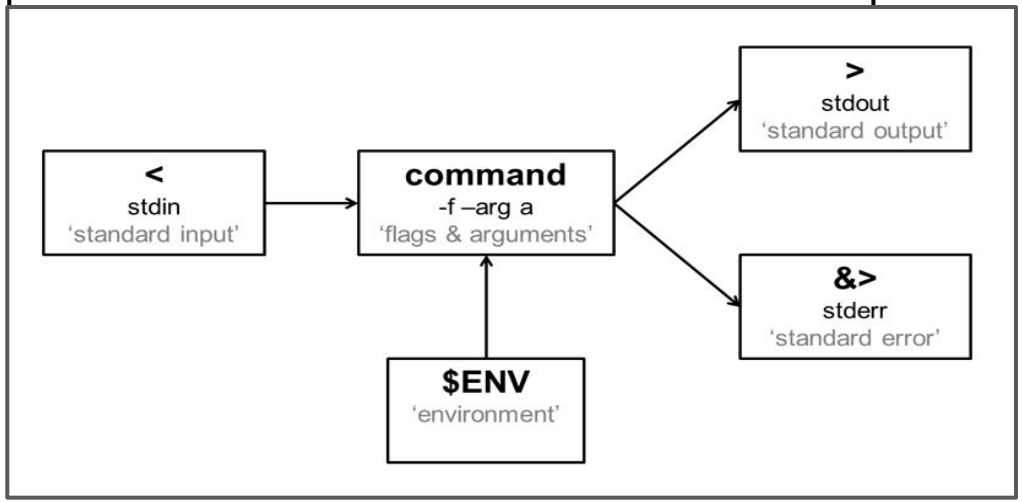
Questions:

1. How many files do we have in the 'data-shell' folder ?
2. How many directories do we have in the 'data-shell' folder ?
3. What does the -r or -R parameter do for most commands 'mv' 'cp' etc?
4. What does the -p option do for the mkdir command ?

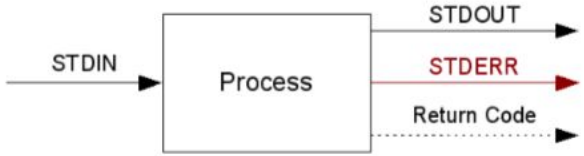
STDIN, STDOUT, STDERR 1/2

Unix processes have some standard ways of handling input and output.
The “environment” is the list of properties the process picks up from its parent. Your processes will all have the shell as their parent.

```
$ tr -d "\t"
$ tr -d "\t" <in >out 2>err
```



STDIN, STDOUT, STDERR 2/2

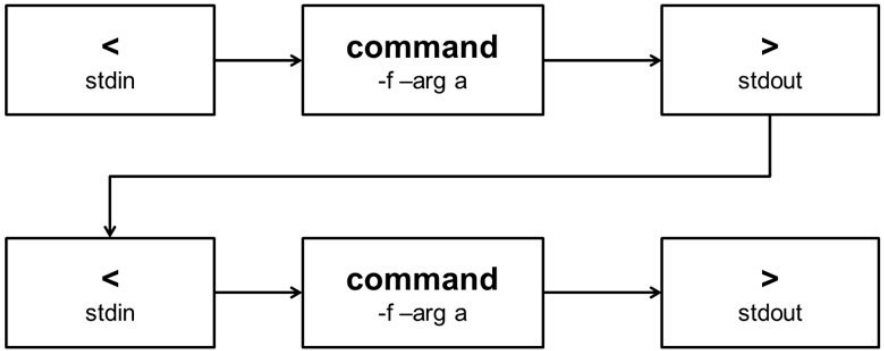


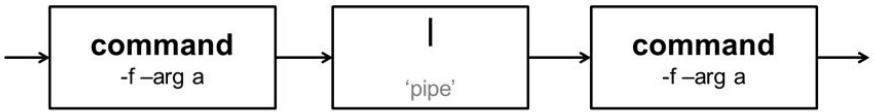

```
>demoprogram <enter>
                                    Hello whats your name?
John <enter>
                                    Hello 'John', how old?
Doe <enter>
                          Error 'Doe' is not a number, how old?
54 <enter>
                                    Hello John you are 54 years old
>echo $? ##this is returncode
```

- **Blue is stdout**
- **Green is stdin**
- **Red is stderr**

STDIN, STDOUT can be chained (PIPES)

- Example. Count the number of files in a directory.
- ls | wc <enter>
  ls is program1
  wc is program2, wordcount
- The vertical line 'l' (not small letter 'l') is called pipe. It glues stdout to stdin
- Output from a process can be written into a file with '>filename'





The 'tr' program is a command that 'translates' characters.

```
$ echo "abcd" >txt.txt
$ cat txt.txt
abcd
$ tr "d" "e" <txt.txt
abce
$ tr "d" "e" <txt.txt >output.txt
$ cat output.txt
abce
$ rm output.txt
$ tr "d" "e" >output.txt <txt.txt
$ cat output.txt
abce
$ cat txt.txt |tr "d" "e"
abce
```

Writing scripts

```
#!/bin/bash

Command1
Command2
Command3
command4
```

#! Is called *shebang* and instructs the shell which program should be used for interpreting the file.

Files containing scripts has to be regular boring textfiles. Not .docx or similar.

sh: bourne shell
csh: C shell
tcsh: TC shell
ksh: Korn shell
bash: bourne again shell
dash: Debian almquist shell
zshell: Default on macOS

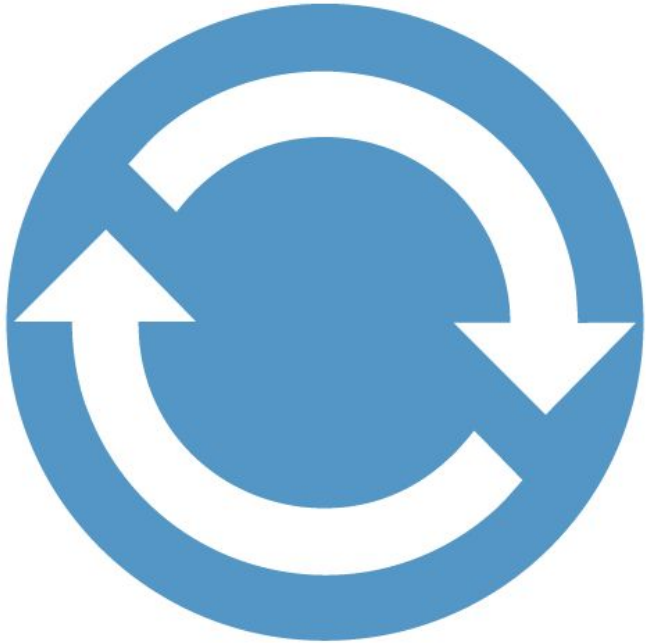
Shell and job control

- Programs can be either terminal programs or graphical programs
- Programs that are started will take focus from terminal until finished, paused (suspended) or put in background
- List of current running process can be found with ‘ps’
- CTRL+C kill program
- CTRL+Z suspend program
- bg=background, fg=foreground
- Programs can be started directly in background by adding ‘&’ after program name

Loops

Loop: A sequence of instructions that is repeated until a certain condition is met

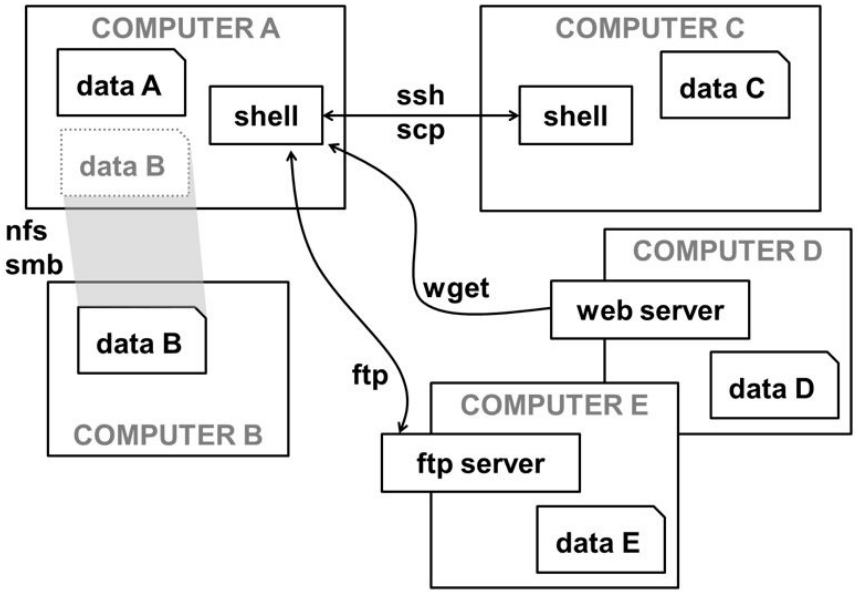
```
for singlefile in $(ls directory/)
do
   head -n4 $singlefile |tail -n2
done
```



- Loops are superusefull since it allows for easy automation
- The block of code that gets repeated is called the loop body
- Each time the loop body is executed is called an iteration

Remote systems?

Most of the ways of moving data around the internet were developed for Unix first. You also have the option of going to where the data is, with a remote shell.



`ssh:` 'secure shell'
Opens up a shell session on a remote machine, over an encrypted channel.

`scp:` 'secure copy'
Carries out a copy between two machines, using the ssh machinery.

`Wget` 'web get'
Lets you grab a file using a url, without all that messing around with web browsers.

`Ftp:` 'file transfer protocol'
Creates another shell-like environment (with a different command set), from which you can connect to other machines and push or retrieve files.

Exercise 2: Software carpentry (shell)

If you have never heard about these concepts before, none of it will have made sense until you have tried it yourself.

Boxes in orange will indicate questions that you should not run but try to resolve by thinking

THE SETUP:

You are ‘Nelle Nemo’, a marine biologist. Your supervisor has given you a great project: but you have to use *her* analysis tools, and they are command line tools that only work on Unix machines...

GO HERE: https://swcarpentry.github.io/shell-novice/

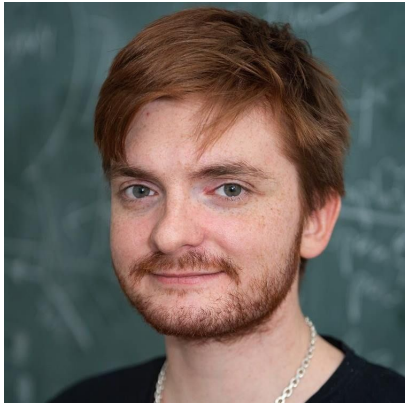
4. Pipes and Filters
5. Loops
6. Shell Scripts

Fundamentals in Computational Analysis of Large-Scale Datasets

8 March 2022

Rasmus Henriksen

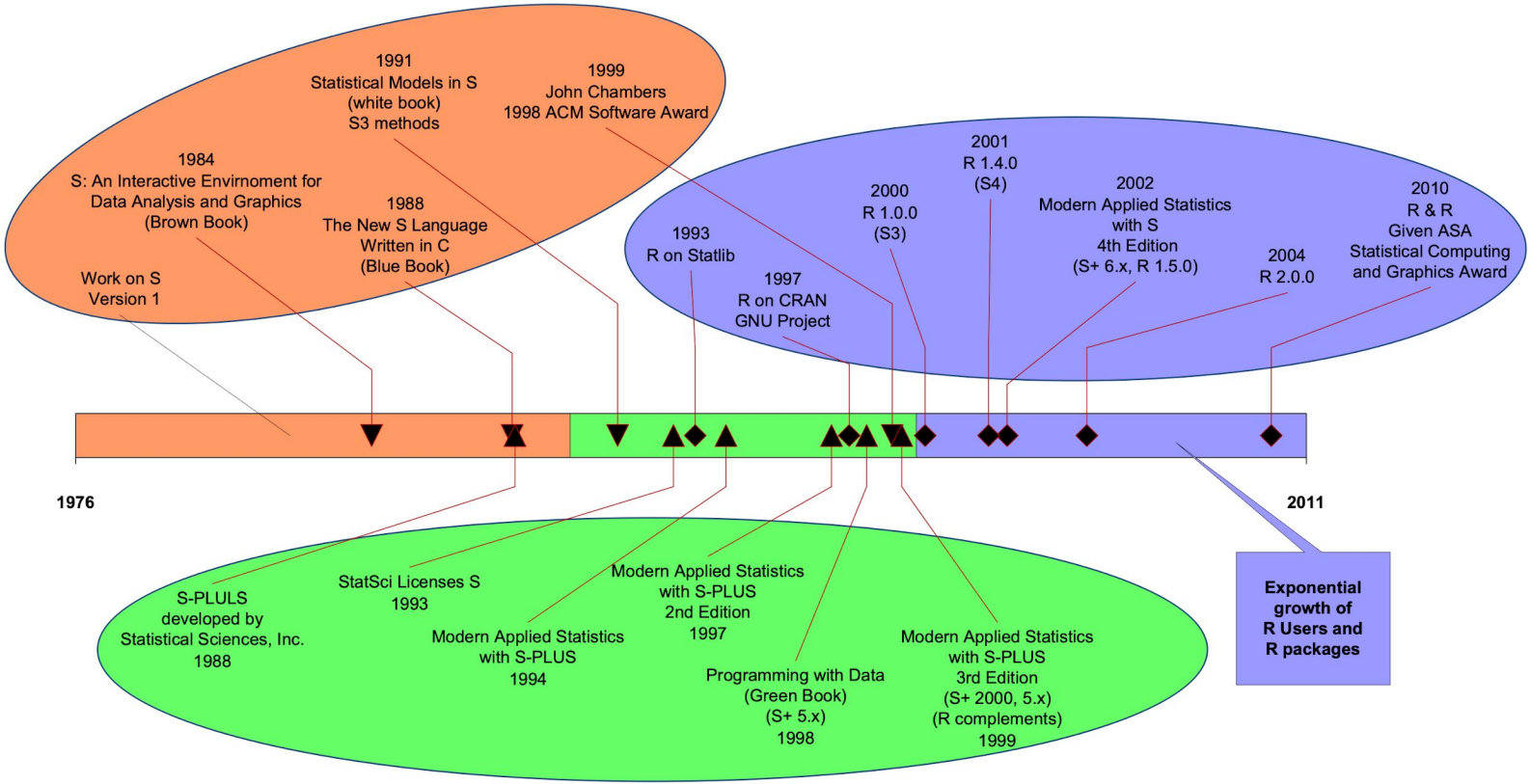
Korneliussen group - Section for GeoGenetics at the GLOBE institute



Rasmus Amund Henriksen

- Msc in Bioinformatics
- PhD in Bioinformatics
- Develop methods to analyse aDNA and NGS data

R programming language



R programming language

- Statistical modelling
- Data analysis
- Data manipulation
- Data visualization
- 9th most popular programming language 2021
- R is an interpreted language:
  - When you enter expressions into the R console or R script
  - A program within the R system, called the interpreter, executes the actual code that you wrote

What is a programming language?

1. Syntax

```
for i in file1 file2
do
        echo ${i}
done
```

2. Semantics

Semantics describe what syntactically valid programs are supposed to do

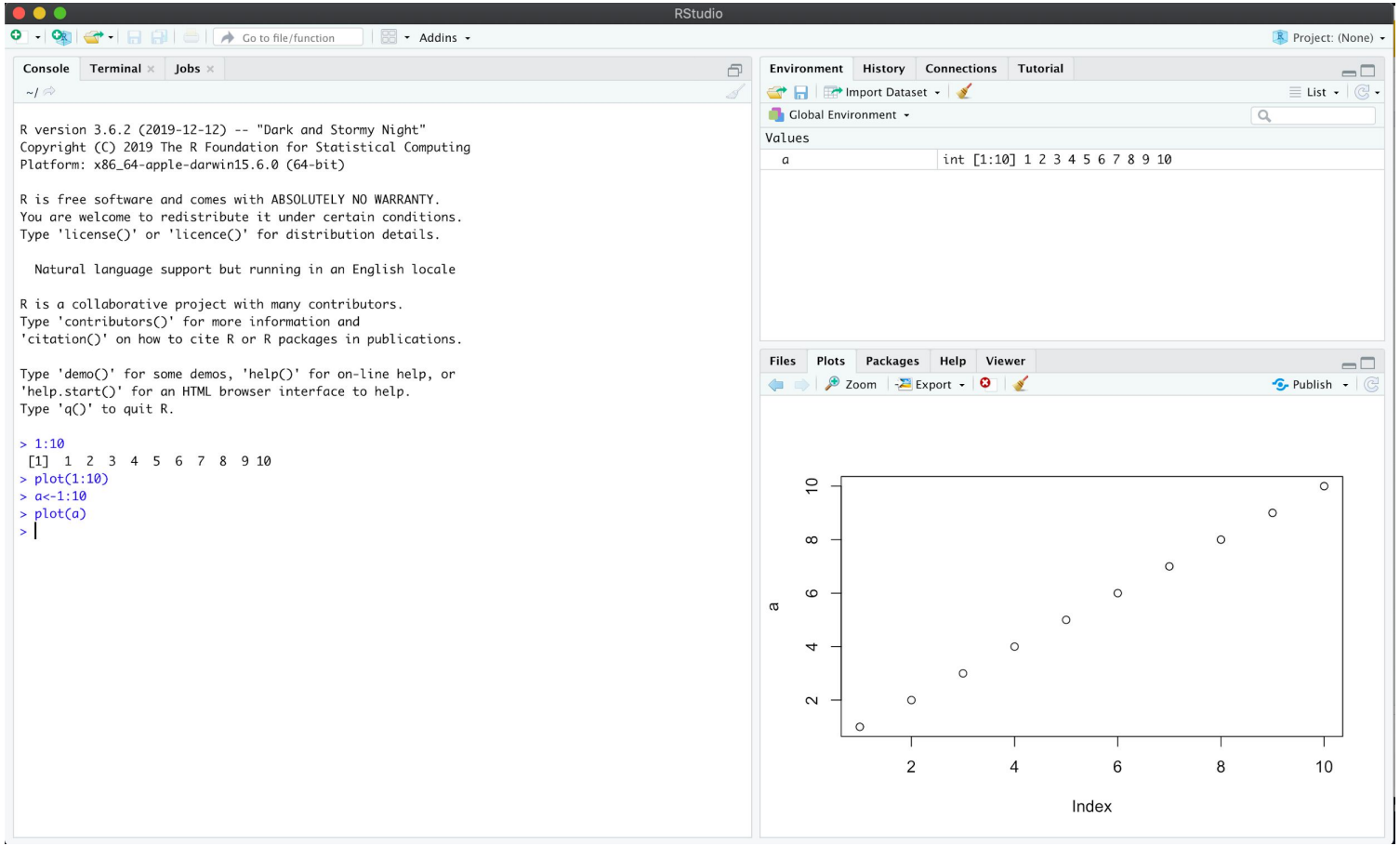
3. Computational model

The computational model describes *how* the output is generated from an input

How to Run R?

1. As a command line script Rscript file.R
2. As an interactive command driven environment
3. As a integrated developer environment RStudio

Rstudio is an integrated developer environment (IDE)



Key Points for march 3

1. Calculator
2. Variables
3. Comments
4. Data types
5. Data structures
6. Builtin functions
7. Writing your own functions
8. Installing packages
9. Simple plotting

Calculator

```
> 2+3+4+5
[1] 14
```

- Red text is what you type in
- Blue text is the response from R

Intended Learning Objectives:

- To be able to open and close R
- To be able to use R as calculator

Variables and comments

1. A variable is a placeholder used to represent a specific value.
2. A comment in R starts with #
3. Arrowsyntax <- General
4. Equalsyntax = used in 'function definitions'
5. Assign() Notice that here the variable name is allowed to be a variable

```
## Calculate area of circle with radius 10
> 3.1415926535897 93*10*10
[1] 314.1593
> pi <- 3.141592653589793
> r <- 10
> pi*r^2
[1] 314.1593
```

R Code: Variable assignment

```
> y <- 5
> y

[1] 5

> assign("e",2.7183)
> e

[1] 2.7183

> s = sqrt(2)
> s

[1] 1.414214

> r <- rnorm(n=2)
> r

[1] 0.4296685 0.4046568
```

R vectors 1/2

Vectors can be constructed with the 'c()' combine function. Used for most operations that work on scalars.

R Code: Creating vectors and vector operations

```
> constants <- c(3.1416,2.7183,1.4142,1.6180)
> names(constants) <- c("pi","euler","sqrt2","golden")
> constants

    pi  euler  sqrt2 golden
3.1416 2.7183 1.4142 1.6180

> constants^2

      pi    euler    sqrt2   golden
9.869651 7.389155 1.999962 2.617924

> 10*constants

    pi  euler  sqrt2 golden
31.416 27.183 14.142 16.180
```

R vectors 2/2

R Code: Indexing vectors

```
> constants[c(1,3,4)]

    pi  sqrt2 golden
3.1416 1.4142 1.6180

> constants[c(-1,-2)]

 sqrt2 golden
1.4142 1.6180

> constants[c("pi","golden")]

    pi golden
3.1416 1.6180

> constants > 2

    pi  euler  sqrt2 golden
  TRUE   TRUE  FALSE  FALSE

> constants[constants > 2]

    pi  euler
3.1416 2.7183
```

```
> constants

    pi  euler  sqrt2 golden
3.1416 2.7183 1.4142 1.6180
```

Vectors are accessed with the bracket notation []
Examples:

1. Positive integer index ranges (extracting)
2. Negative integer index ranges (discarding)
3. By names
4. By logical vector

Datatypes 1/2

1. Everything in R is on object and all objects has classes
2. Use ls() or objects() functions to obtain a character vector of the names of the current existing objects
3. Use the class() function to obtain the class of the specified object
4. A *datatype* is an attribute of some data that tells the interpreter what the type is.
5. R has six *atomic* datatypes (these can be adjoint to a *vector*)

typeof	mode	storage.mode
logical	logical	logical
integer	numeric	integer
double	numeric	double
complex	complex	complex
character	character	character
raw	raw	raw

Classical classes

- Numeric
- Character
- Data.frame
- Matrix

Datatypes 2/2

```
> x <- c(3.1416,2.7183)
> m <- matrix(rnorm(9),nrow=3)
> tab <- data.frame(store=c("downtown","eastside","airport"),sales=c(32,17,24))
> cities <- c("Seattle","Portland","San Francisco")
```

```
> ls()
[1] "cities" "m"      "tab"    "x"
> objects()
[1] "cities" "m"      "tab"    "x"
> class(m)
[1] "matrix"
> class(x)
[1] "numeric"
```

R Code: Object type (storage mode)

```
> x
[1] 3.1416 2.7183
> typeof(x)
[1] "double"
> cities
[1] "Seattle"       "Portland"
[3] "San Francisco"
> typeof(cities)
[1] "character"
```

Data.frame vs Matrix

Matrix:

- Two-dimensional - m*n
- Similar datatypes (homogeneous)
- Fixed number of rows and columns

Data.frame:

- Generalized form of matrix
- Different data types (heterogeneous)
- Variable number of rows and columns

R Code: Object class

```
> m
           [,1]       [,2]      [,3]
[1,] -0.6147361 -0.2248133 0.1354078
[2,] -0.7835507  2.3798959 0.8825350
[3,]  1.0156090  1.4605885 0.9470563

> class(m)

[1] "matrix"

> tab

     store sales
1 downtown    32
2 eastside    17
3  airport    24

> class(tab)

[1] "data.frame"
```

Data.frame - useful functions

```
> df<-read.table(file="file.csv" , sep="," , skip=1 )

>write.table(df,"dfSaved.txt",sep="\t",row.names=FALSE,col.names=TRUE)
```

Slice and Subset of dataframes

row1 is `df[1,]` row2 is `df[2,]`

column1 is `df[,1]` column2 is `df[,2]`

Rename dataframes

colnames(df) <- c(“Long”, ”Lat”, ”City”)

Bind dataframes

rbind(`df[,1],df[,2]`)

`cbind(df[,2],df[,1],df[,1])`

`cbind(df$Long,df$City)`

```
#longitude,latitude,city name
145.768,-16.915,"Cairns"
146.801,-19.265,"Townsville"
150.501,-23.365,"Rockhampton"
139.485,-20.715,"Mount Isa"
150.893,-34.423,"Wollongong"
151.785,-32.932,"Newcastle"
141.451,-31.965,"Broken Hill"
145.951,-30.082,"Bourke"
150.932,-31.091,"Tamworth"
149.581,-33.417,"Bathurst"
153.118,-30.315,"Coffs Harbour"
146.901,-36.065,"Albury"
```

Functions (builtin)

We can sample numbers from a normal (gaussian) distribution with the rnorm function.

```
> args(rnorm)
function (x, mean = 0, sd = 1)
NULL
```

- This function takes 3 arguments.
- 2 of the arguments has default values

```
> rnorm(2)
[1] -0.05526951  1.03702853
> rnorm(2,sd=10)
[1] -10.487585   3.841609
> rnorm(2,sd=10,mean=100)
[1] 108.5422 126.7425
```

1. Unnamed arguments are assigned according to position
2. Named argument are assigned by name (partial matching)
3. Only arguments with no default value are required to be supplied

Writing your own functions

Its very easy to write your own functions

```
> myadder <- function(a,b=2){
        mysum <- a+b
        return(mysum)
}
> myadder(2)
[1] 4
>myadder(2,b=3)
[1] 5
```

- This function takes 2 arguments.
- Second argument is not mandatory

The arguments for a function in R can in itself be a function.

Variables that are defined *within* a function does not exist after the function has run.

You can define functions *within* a function, then these will not be visible outside the function.

Installing packages

- All functions in R are stored in packages
- Most usefull functions are included in core packages which is included in a standard R installation

You can develop your own Rscripts that parses and analyses some data. But sometimes you need some functionality that is not included in a standard installation of R.

R Code: The `install.packages` function

```
> args(install.packages)

function (pkgs, lib, repos = getOption("repos"), contriburl = contrib.url(repos,
    type), method, available = NULL, destdir = NULL, dependencies = NA,
    type = getOption("pkgType"), configure.args = getOption("configure.args"),
    configure.vars = getOption("configure.vars"), clean = FALSE,
    Ncpus = getOption("Ncpus", 1L), libs_only = FALSE, INSTALL_opts,
    ...)
NULL

> #install.packages("nutshell")
> # or if repos needs to be specified
> #install.packages("nutshell", repos="http://cran.fhcrc.org")
```

After you have installed the package you will need to "load" it

> library("nutshell")

Plotting data 1/2

- R is very useful as a calculator
- R is very useful for data exploration and manipulation
- R is very useful for visualizing data and making publication ready figures

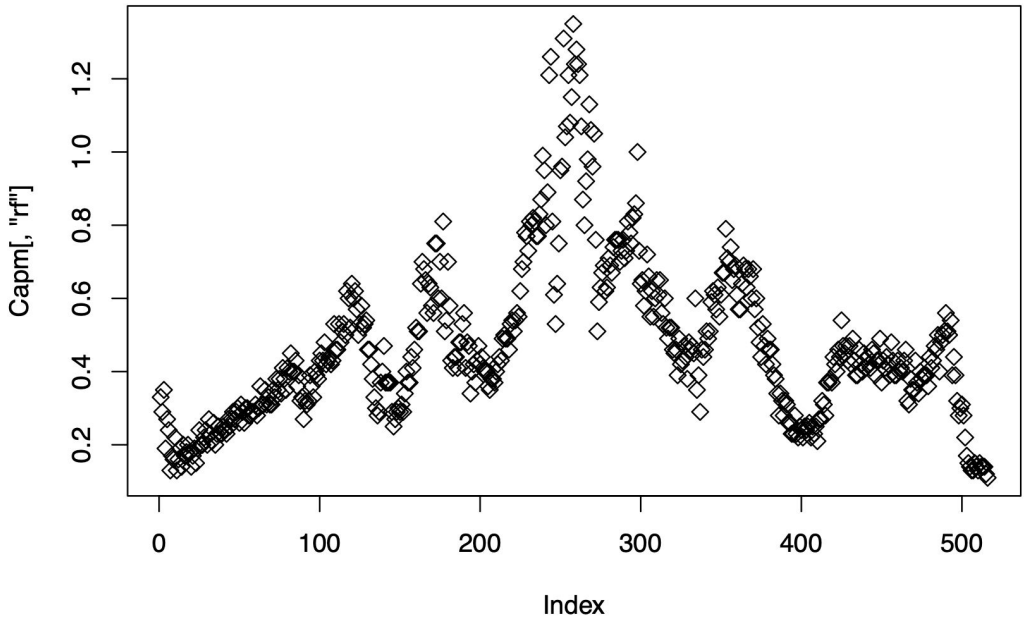
Below are some of the most important plotting functions

Function	Description
`plot`	generic function to plot an R object
`lines`	adds lines to the current plot
`segments`	adds lines line segments between point pairs
`points`	adds points to the current plot
`text`	adds text to the current plot
`abline`	adds straight lines to the current plot
`curve`	plot a function over a range
`legend`	adds a legend to the current plot
`matplot`	plot all columns of a matrix
`par`	sets graphics parameters

Plotting data 2/2

R Code: Plot with defaults

```
> library(Ecdat)
> data(Capm)
> plot(Capm[,"rf"],pch=5)
```



Important topics that has not been covered

1. The data structure called **list()**
2. Writing data to files **write.table()**
3. Factor levels
4. Writing figures to files **pdf(),png()** etc

Some of the examples are based on the following slideshow:
https://faculty.washington.edu/ezivot/econ424/RIntro.pdf