Introduction to R and UNIX
Working with microarray data in a multi-user environment

Carsten Friis
Introduction to UNIX

Working in a multi-user environment
What do you need to know?

This is not a course on computers

But you will need some UNIX for the exercises, and for your final project

You will also need to know some R to handle the exercises and for your final project

And you will be expected to perform analysis in R during your evaluated projects
What is UNIX?

UNIX is not just one operating system, but a collection of many different systems sharing a common interface.

It has evolved alongside the internet and is geared toward multi-user environments, and big multi-processors like our servers.

At its heart, UNIX is a text-based operating system – that means a little typing is required.

UNIX Lives! (Particularly Linux and MAC OS X!)
UNIX comes with a help system

It depends on the program in question:

– Either as a ‘man-page’ manual page using the command ‘man’
  • Example: man less

– Or through the ‘-h’ option
  • Example: acroread ‘-h’

By the way, in the ‘man-pages’ you scroll with ‘k’ and ‘j’ 😊
Navigating Directories

Key commands:
- `ls` #lists the files in the current directory
- `cd dir` #changes working directory to ‘dir’
- `mkdir dir` #makes a new directory called ‘dir’

Nice tricks:
- The shorthand ‘ll’ is short for `ls -l`
- The asterisk ‘*’ is a wildcard – ‘ls *.txt’ will list all files ending with ‘.txt’
- ‘cd ..’ takes you back one directory
- plain ‘cd’ (no arguments) takes you to your home directory
Basic File Handling

Key commands:
- `cp x y`  #creates a copy of file ‘x’ named ‘y’
- `mv x y`  #moves file ‘x’ to file ‘y’. File ‘x’ is erased
- `rm x`  #removes (erases) file ‘x’. (Tip: use `ls x` first!)
- `ln –s x y` #creates a symbolic link to ‘x’ called ‘y’. Similar to shortcuts in Windows

Nice tricks:
- A lone period ‘.’ means current directory. This means you can copy a file into current dir like this ‘`cp elsewhere/file .`’
- The ‘-r’ option to `rm` removes directories. e.g. ‘`rm –r dir`’
- Many commands have options accessed using a ‘-’ Read the man-pages to see all options
Handling text (i.e. data) files

Key commands:

- `wc`  #Size of file in lines, words and characters
- `nedit`  #Text editor for manipulating text files
- `less`  #Views text files
- `cut`  #Extracts columns from tabular text files
- `grep`  #Finds lines in files with specified keywords

Use man or ‘-h’ for more info on any of these...
UNIX: Moving files...

In the ssh program under Windows, look for an icon resembling a yellow directory folder with blue dots on.

If you use putty, you need the separate ‘psftp’ program, which is not that user-friendly.
Can you see the papers between those spots?
Working with R

(...and other hazardous activities)
Literature on R

Documentation used in this course can be found on the course webpage, it includes:

- Beginner's Guide to UNIX
- These lecture notes (hopefully 😊)
- An Introduction to R

Many good R manuals for further reading can be found on the web:

- http://cran.r-project.org/manuals.html
What is R?

It began with ‘S’. ‘S’ is a statistical tool developed back in the 70s.

R was introduced as a free implementation of ‘S’. The two are still quite similar.

R is freeware under the GNU license, and is developed by a large net of contributors.
Why use R? (And not Excel?)
Mistaken Identifiers: Gene name errors can be introduced inadvertently when using Excel in bioinformatics

Barry R Zeeberg, Joseph Riss, David W Kane, Kimberly J Bussey, Edward Uchio, W Marston Linehan, J Carl Barrett and John N Weinstein

Background: When processing microarray data sets, we recently noticed that some gene names were being changed inadvertently to non-gene names.

Results: A little detective work traced the problem to default date format conversions and floating-point format conversions in the very useful Excel program package. The date conversions affect at least 30 gene names; the floating-point conversions affect at least 2,000 if Riken identifiers are included. These conversions are irreversible; the original gene names cannot be recovered.

Conclusions: Users of Excel for analyses involving gene names should be aware of this problem, which can cause genes, including medically important ones, to be lost from view and which has contaminated even carefully curated public databases. We provide work-arounds and scripts for circumventing the problem.
LocusLink Screenshot (Zeeberg et al. 2004)
Why use R? (And not Excel?)

R has specific functions for bioinformatics in general, and for microarrays in particular.

R is available for (almost) all platforms – e.g. Linux, MacOS, Win32/WinXP

The R community is quite strong, and updates appear regularly

Oh, and R happens to be freeware...
Starting with R

Just type ‘R’ on the command line
  – Or ‘R-2.6.1’, to ensure you get a recent version

How to get help:

  > help.start()     #Opens browser
  > help()           #For more on using help
  > help(.)          #For help on ..
  > help.search("..")  #To search for ..

How to leave again:

  > q()              #Image can be saved to .RData
Basic R commands

Most arithmetic operators work like you would expect in R:

\[
\begin{align*}
&> 4 + 2 \quad \#\text{Prints ‘6’} \\
&> 3 \times 4 \quad \#\text{Prints ‘12’}
\end{align*}
\]

Operators have *precedence* as known from basic algebra:

\[
\begin{align*}
&> 1 + 2 \times 4 \quad \#\text{Prints ‘9’, while} \\
&>(1 + 2) \times 4 \quad \#\text{Prints ‘12’}
\end{align*}
\]
A function call in R looks like this:
- `function_name(arguments)`
- Examples:
  
  ```
  > cos(pi/3)  #Prints `0.5`
  > exp(1)     #Prints `2.718282`
  ```

A function is identified in R by the parentheses
- That’s why it’s: `help()`, and not: `help`
To assign a value to a variable (object):

```r
> x <- 4         # Assigns 4 to x
> x = 4          # Assigns 4 to x (new)
> x              # Prints ‘4’
> y <- x + 2     # Assigns 6 to y
```

Functions for managing variables:
- `ls()` or `objects()` lists all existing objects
- `str(x)` tells the structure (type) of object ‘x’
- `rm(x)` removes (deletes) the object ‘x’
Vectors in R

A vector in R is like a sequence of elements of the same mode.

```r
>x <- 1:10          #Creates a vector
>y <- c(“a”, “b”, “c”)  #So does this
```

Handy functions for vectors:
- `c()` – Concatenates arguments into a vector
- `min()` – Returns the smallest value in vector
- `max()` – Returns the largest value in vector
- `mean()` – Returns the mean of the vector
R is prepared to handle objects with large amounts of data.

All simple numerical objects in R function like a long string of numbers.

In fact, even the simple: \( x \leftarrow 1 \), can be thought of like a vector with one element.

The functions \texttt{dim(x)} and \texttt{str(x)} returns information on the dimensionality of \( x \).
Important object types in R

vector – “A series of numbers”

matrix – “Tables of numbers”

data.frame – “More ‘powerful’ matrix (list of vectors)”

list – “Collections of other objects”

class – “Intelligent(?) lists, holds objects and functions”
More on Vectors

Elements in a vector can be accessed individually:

> x[1]       #Prints first element
> x[1:10]    #Prints first 10 elements
> x[c(1,3)]  #Prints element 1 and 3

Most functions expect one vector as argument, rather than individual numbers

> mean(1,2,3)       #Replies ‘1’
> mean(c(1,2,3))    #Replies ‘2’
Conditional indexes and Booleans

You can use conditionals as indexes!

\[
\begin{align*}
&x \leftarrow 1:10 \ #\text{Data to play with} \\
&x > 5 \ #\text{Prints Boolean vector} \\
&x[x > 5] \ #\text{Elements greater than 5}
\end{align*}
\]

You can use the `which()` function for even greater control, and the `grep()` function to search for complete or partial strings.
The Recycling Rule

The *recycling rule* is a key concept for vector algebra in R.

When a vector is too short for a given operation, the elements are recycled and used again.

Examples of vectors that are too short:

```r
>x <- c(1,2,3,4)
> y <- c(1,2)  # y is too short
> x + y       # Returns '2,4,4,6'
```
Data Matrices

Matrices are created with the `matrix()` function.

```r
> m <- matrix(1:12, nrow=3)
```

This produces something like this:

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>4</td>
<td>7</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>5</td>
<td>8</td>
<td>11</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>6</td>
<td>9</td>
<td>12</td>
</tr>
</tbody>
</table>
Matrices also recycle

The *recycling rule* still applies:

```r
> m <- matrix(c(2,5), nrow=3, ncol=3)
```

Gives the following matrix:

```
[,1] [,2] [,3]
[1,]  2  5  2
[2,]  5  2  5
[3,]  2  5  2
```
Indexing Matrices

For vectors we could specify one index vector like this:

```r
>x <- c(2,0,1,5)
>x[c(1,3)]    #Returns ‘2’ and ‘1’
```

For matrices we have to specify two vectors:

```r
>m <- matrix(1:3,nrow=3,ncol=3)
>m[c(1,3),c(1,3)]  #Ret. 2*2 matrix
>m[1,]         #First row as vector
```
The **apply** function

The `apply()` function ‘applies’ another function along a specified dimension.

- `apply(m,1,sum)` #Sum of rows
- `apply(m,2,sum)` #Sum of columns
- `apply(m,1,mean)` #Mean of rows

Apply has cousins:

- Like `lapply()` which works for *list* objects.
Beyond two Dimensions

You can actually assign to `dim()`:

```r
> x <- 1:12
> dim(x)             #Returns ‘NULL’
> dim(x) <- c(3,4)   #3*4 Matrix
> dim(x)             #Returns ‘3 4’
> dim(x) <- c(2,3,2) #x is now in 3d
> dim(x)             #Returns ‘2 3 2’
```

But functions like `mean()` still work:

```r
> mean(x)            #Returns ‘6.5’
```
List Objects

A list is like a vector, but it can store *anything*, even other objects!

Lists are created with the `list()` function:

```r
-> l1 <- list(1:10, c("a","b"))
-> l1[[2]]          #Returns ‘a b’

-> l2 <- list(a=1:10, b=c("a","b"))
-> l2$b             #Returns ‘a b’
```
Data frames work like a list of vectors of identical length.

Data frames are created like this:

\[- f \leftarrow \text{data.frame}(\text{cbind}(x=1,y=1:5))\]

The demonstration dataset USArrests is a data.frame.

\[- > \text{data(USArrests)} \ #\text{Activate dataset}\]
\[- > \text{str(USArrests)} \ #\text{View the structure}\]
Data frames work like Matrices (But lists do NOT!)

Standard algebra, add to each element:

\[ \rightarrow \text{USArrests} \ + 100 \]

Indexing data frames:

\[ \rightarrow \text{USArrests}[2,] \quad \#\text{Stats for Alaska} \]

Average arrests across the States:

\[ \rightarrow \text{apply}(\text{USArrests},2,\text{mean}) \]
Visualization is one of R’s strong points.

R has many functions for drawing graphs, including:
- `hist(x)` – Draws a histogram of values in x
- `plot(x, y)` – Draws a basic xy plot of x against y

Adding stuff to plots
- `points(x, y)` – Add point (x,y) to existing graph.
- `lines(x, y)` – Connect points with line.
- `text(x, y, str)` – Writes string at (x,y).
Graphical Devices in R

A graphical device is what ‘displays’ the graph. It can be a window, it can be the printer.

Functions for plotting “Devices”:

- `x11()` – This function allows you to change the size and composition of the plotting window.
- `par(mfrow=c(x, y))` – Splits a plotting device into x rows and y columns.
- `dev.copy2eps(file="????.ps")` – Use this function to copy the active device to a file.
Exercises in R

To warm you up, open the “Basic R Commands” exercise on the course webpage
– When finished, feel free to play with some more demos
  • type “demo()” to see what’s available

[Optional] Proceed with the extra exercise:
– These exercises are hard! (that’s why they are optional)