Coursework_1_6BBG0310_k20041606

Jude Popham, *K20041606*

2022-11-04

Section 1: UNIX

Question U1

Question: List five types of unix command and describe what each command does, using an example in each case.

- **Answer 1**: '*pwd*'
- **Explanation**: pwd means 'print working directory'. This is an essential command to find out where you are located within your files. The file system in a computer is the directory tree and the base of this tree is the root directory: /. A directory is essentially a folder and the tree consists of folders wtihin folders.
- **Example**:

```
#Putting the directory as the folder which contains this coursework
cd Documents/Bioinformatics_Module/CW_1/Markdown/
#Using pwd to show/check that this is the current working directory
pwd
```

- **Answer 2**: '*cd*'
- **Explanation**: cd means 'change directory'. This is an essential command to change location/directory within the directory tree of the computer. You can begin with the root directory, /, and then go up the 'directory tree' and enter folders within folders by stating their names followed by a /.
- **Example**:

```
#We start in the folder of our coursework
pwd
#We change directory to the folder of the data which contains this coursework.
#We go back using .. and then enter the Data folder using /Data
cd ../Data/
```

- **Answer 3**: '*nano*'
- **Explanation**: nano is a command that can be used to open a text editor which can be used to create a text file. For example if we want to create a text file saying 'hello world' we would use the code below:
- **Example**:

```
#First we change the directory to a place we want the file
cd ../Markdown/
#First we write nano in the terminal
nano
#Then we write the contents of our file in terminal: 'hello world' and
#'good night world' on the next line
#Then we press control x to finish editing
#Then we press Y for yes to save the file in the directory we were in
#when we used nano and write the name of the file, with .txt at the end
#Now we can check the file within the directory it was created using 'cat'
cat hello_world.txt
```

- **Answer 4**: '*mv*'
- **Explanation**: mv is a command that can be used to move files bewteen directories. The basic order is mv (file name) (new place)
- **Example**:

```
#Let's check the directory containing our new 'hello_world' text file we created
#in the previous step
pwd
#Let's now move this file into a different directory (Data) within the directory
#that contains the Markdown directory
mv hello_world.txt ../Data
```

- **Answer 5**: '*sed*'
- **Explanation**: sed is a command which means 'stream editor'. It is used to replace strings with new strings and delete lines in text files.
- **Example**:

```
###Substituting with sed
##Let's re-open the directory containing the hello_world.txt file
cd ~
cd Documents/Bioinformatics_Module/CW_1/Data
##Let's substitute the word 'night' with 'morning' in the sentence
#'good night world' in line 2
sed 's/night/morning/g' hello_world.txt > hello_world_morning.txt
##Deleting the second line of the file with sed and overwriting the file
#using sed -i.bak
#The i.bak is needed specifically rather than sed -i in macOSX
sed -i.bak '2d' hello_world.txt
#Checking the file has removed the second line so hello_world.txt is now just
#hello world
cat hello_world.txt
```

Question U2

Question: Within the data directory, the second largest file contains information on genetic variants that have been identified for a group of human individuals. A collaborator is interested in information about how the data was generated (contained within the header of the file, lines that start with #), and also wants to see some examples of genetic variant calls. Identify this file from the directory and produce a new file from this that contains the header and information on the last five genetic variants. Move this file to a new directory that you call 'CollaboratorA'.

- The **first** step is to enter the data directory using the *cd* command.
- The **second** step is to identify the second largest file using the *ls -S* command which will sort all of the files in order of size.
- The **third** step would be to extract header information using the # wildcard into a new file and put this into a new file called Data_Generation.
- The **fourth** step would be to find the last 5 genetic variants and append this information to the new file, called Data_Generation.
- The **fifth** step would be to make a new CollaboratorA directory via *mkdir* and move this file into it with the *mv* command.

Code:

```
cd /Users/judepops/Documents/Bioinformatics_Module/CW_1/Data
ls -S
grep -E "^.#" ALL.integrated_snvindels_v2a_27022019.GRCh38.phased.vcf > Data_Generation
tail -5 ALL.integrated_snvindels_v2a_27022019.GRCh38.phased.vcf >> Data_Generation
mkdir CollaboratorA
mv Data_Generation CollaboratorA
```

Output:

```
cd /Users/judepops/Documents/Bioinformatics_Module/CW_1/Data/CollaboratorA
cat Data_Generation
```

Question U3

Question: There are three text files labelled "Group.txt" that you wish to share with a collaborator as a single file, but you will have to remove the date of birth information, as this is personal information and you do not have 1 permission to share it. Individual 14 has also withdrawn from the study, so you should not share data for this individual. Create this file, maintaining a single header at the top and including all permitted individuals across the three groups.

- The **first** step is to remove headers from all the group.txt files and create individual files
- The **second** step is to append these individual files to eachother = Groups.txt
- The **third** step is to remove indidviual 14 using the *sed* command.
- The **fourth** step is to remove the dates using the command: *cut*.
- The **fifth** step is to remove the dates using the command: *sed*. We have to substitute (/s) using sed because using the /d delete the whole line. The dates pattern can be created using a ../../.... representation.
- The **sixth** step is to remove the DOB string within the header using the command: *sed*.

Code:

```
cd /Users/judepops/Documents/Bioinformatics_Module/CW_1/Data

for G in Group*.txt
do grep -vwE "DOB" "$G" > $G.txt
done

cat GroupA.txt.txt GroupB.txt.txt GroupC.txt.txt > Groups.txt
head -n 1 GroupA.txt > Groups_Header.txt
cat Groups.txt >> Groups_Header.txt
sed '/^14/d' Groups_Header.txt
sed -i "" "s|../../....||g" Groups_List.txt
sed -i "" "s|DOB||g" Groups_List.txt
```

Output:

```
cd /Users/judepops/Documents/Bioinformatics_Module/CW_1/Data
cat Groups_List.txt
```

Question U4

Question: Within the directory there is a ".sam" file containing information from a sequencing experiment in SAM format. The first column of a SAM file contains the read name and subsequent columns give information about where the read aligns to a reference genome. Each sequencing read can align to multiple different locations and thus have multiple different entries (on different lines) in the SAM file. Count the total number of lines in the file, and then count the number of unique read names in the file (so if a read name occurs more than once on different lines, you should only count it once).

- The **first** step is to count the total number of lines in the file using the *wc -l* command = *17608*.
- The **second** step is to use the tab delimiter with the *cut* command to remove columns: this is possible because a .sam file is a form of .tsv file so it is a tab delimited text file. Fortunately, this is the default delimiter for the cut command so the command to use is simply: *cut -f 1*.
- The **third** step is to remove duplicates and count the number of lines. This can be done with a combination of the *sort* and *uniq* commands. This automatically sorts and removes duplicates when the two commands are used in a pipe. This gives us an answer of **8804** unique lines.

Code:

```
cd /Users/judepops/Documents/Bioinformatics_Module/CW_1/Data
#Working out file length
wc -l HG01334.sam
#Extracting the first column ('read names') into a new file
cut -f 1 > HG01334_readnames.txt
#Removing duplicates
sort HG01334_readnames.txt | uniq > HG01334_readnames_unique.txt
#Counting the number of lines again
wc -l HG01334_readnames_unique.txt
```

Output:

```
cd /Users/judepops/Documents/Bioinformatics_Module/CW_1/Data
wc -l HG01334_readnames_unique.txt
```

Question U5

Question: A researcher in your group has collected some information about different genes by merging different datasets, but has made some errors in their code. First, when matching gene names from different datasets, the wrong information has been collected in some cases so gene names in columns 2 and 3 do not always match (these lines need to be removed). Second, the word 'Ortholog' was mistyped as 'Ortholag', meaning that downstream searches for orthologous genes would not work for all genes. Using command line arguments, go through the file "genes.information.csv" (columns are separated by commas) and correct these two errors, creating a new file called "genes.information.corrected.csv" that contains the original header as the first line.

- The **first** step is to remove lines where columns 2 and 3 do not match. In this case, we are using a csv file so all columns are separated by commas as the primary delimiter.
- The **second** step is to substitute Ortholag for Ortholog. This can be done using the *awk -F* command.*Awk* processes data by scanning patterns and the -F specifies the delimiter as , automatically. Here we extract columns where column 2 is equal to column 3.

Code:

```
cd /Users/judepops/Documents/Bioinformatics_Module/CW_1/Data
#Replacing Ortholag with Ortholog
sed -i "Ortholog" "s|Ortholag||g" genes.information.csv
#Identifying and removing lines where columns 2 and 3 do not match and save this
#as a new file.
awk -F, '$2==$3' genes.information2.csv > genes.information.corrected.csv
```

Output:

```
cd /Users/judepops/Documents/Bioinformatics_Module/CW_1/Data
#Printing the output file
cat genes.information.corrected.csv
```

Section 2: R

Question R1:

Question: Define the vector x <- c(5,9,2,3,4,6,7,0,8,12,2,9). For each of the following questions, you should use just a single line of code in R to calculate the answer: a) What is the sum of the first 4 elements? b) What is the highest value in the vector? c) Which position in the vector contains the highest value? d) Which position in the vector would the highest value occur if the vector is reversed in order? e) How many values in the vector are lower than 10, but higher than 5?

- **a**: 19
- **b**: 12
- **c**: Position 10
- **d**: Position 3
- **e**: 5

Code:

```
#Libarires
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
#Creating the variable
x <- c(5,9,2,3,4,6,7,0,8,12,2,9)
#a
sum(x[1:4])
```

```
## [1] 19
```

```
#b
max(x)
```

```
## [1] 12
```

```
#c
which.max(x)
```

```
## [1] 10
```

```
#d
rev(x) %>% which.max()
```

```
## [1] 3
```

```
#e
sum(x<10 & x>5)
```

```
## [1] 5
```

Question R2:

Question: There are 500 rooms in a building. Room 1 contains 1 box, room 2 contains 2 boxes and so on. Each box contains two items. Calculate how many items there are in total. If there were only 400 rooms in the building, with two items in each box, how many fewer items would there be in this building compared to the first?

- **Items in total**: 250500
- **Fewer items in this building compared to the first**: 45050

Code:

```r
#Working out the number of boxes for 500 rooms
sum_boxes_500 <- 0
for (i in 1:500) {
  sum_boxes_500 <- sum_boxes_500 + i
}
#Working out the number of items for 500 rooms
sum_items_500 = 2*(sum_boxes_500)

#Working out the number in 400 rooms
sum_boxes_400 <- 0
for (i in 1:400) {
  sum_boxes_400 <- sum_boxes_400 + i
}

#Working out the number of itens 400
sum_items_400 <- 2*(sum_boxes_400)

#Working out difference in items (how many fewer in building 2)
fewer_items_in_building_2 <- sum_boxes_500 - sum_boxes_400
print(fewer_items_in_building_2)
```

```
## [1] 45050
```

Question R3:

Question: For this question, use basic R commands. The file 'cell_features.csv' contains information from different cell lines and experiments that have been run in different fibronectin concentrations. Create a new data structure containing cell area means from batch A only and then calculate the median value and interquartile range of these.

- **New data structure**: *cell_features_A*
- **Answer 1**: Median = 3.135937
- **Answer 2**: IQR = 0.2272149

```r
#Loading dataframe
cell_features <- read.csv('/Users/judepops/Documents/Bioinformatics_Module/CW_1/Data/cell_features.csv'
#Creating dataframe of cell area means from batch A
```

```
cell_features_A <- cell_features %>%
  filter(batch == 'Batch A') %>%
  select(cell_area_mean)
#Calculating IQR and median and saving these values in variables
cell_features_A_median <- median(cell_features_A$cell_area_mean)
cell_features_A_IQR <- IQR(cell_features_A$cell_area_mean)
```

Question R4:

Question: The file 'humans.csv' contains information for ten individuals, detailing their sex, whether they are male, their age and their favourite colour. Load the file into R as an object, view it and correct any errors you observe using R directly. Calculate the mean age of individuals in the file - if you get any error messages along the way, try to work out why, and make the necessary changes to the object so that all future operations using the object will work as intended.

- **Corrected Dataframe**: *humans_data*
- **Answer**: Mean age individuals = 35.9

Code

```
#Loading the dataframe
humans_data <- read.csv('/Users/judepops/Documents/Bioinformatics_Module/CW_1/Data/humans.csv')
#Correcting the dataframe and saving it as humans_data_corrected
humans_data[which(humans_data$Is_Male == 29), c("Is_Male", "Age")] <- rev(humans_data[which(humans_data$
#Checking classes of columns
sapply(humans_data, class)
```

```
##      Sample         Sex     Is_Male         Age      colour
##   "integer" "character" "character" "character" "character"
```

```
#Converting column Age into an integer from character
humans_data <- transform(humans_data, Age = as.numeric(Age))
#Working out the mean
mean(humans_data$Age)
```

```
## [1] 35.9
```

Question R5:

Question: COVID-19 Data for the UK is hosted on the website https://coronavirus.data.gov.uk. Go to this site, browse the data and instructions and work out how to download the data directly into R (Not via a spreadsheet -

Hint: Use the "Developer's Guide" information). Find the date in the last 500 days that had the highest number of cases. Comparing the latest figures to those 300 days ago, have COVID-19 case numbers gone up or down?

- **Date in last 500 days with highest number of cases**: 2022-03-21
- **COVID cases up or down compared to 300 days ago?**: Down

Code

```
#Loading the data into R - Pillar 1 and 2 as this is swab testing for the wider population
library(readr)
Coronavirus_Data <- read.csv('https://api.coronavirus.data.gov.uk/v2/data?areaType=overview&metric=newC
head(Coronavirus_Data)
```

```
##    areaCode       areaName areaType       date newCasesByPublishDate
## 1 K02000001 United Kingdom overview 2022-05-20                  6338
## 2 K02000001 United Kingdom overview 2022-05-19                 12208
## 3 K02000001 United Kingdom overview 2022-05-18                  7791
## 4 K02000001 United Kingdom overview 2022-05-17                  8603
## 5 K02000001 United Kingdom overview 2022-05-16                 22337
## 6 K02000001 United Kingdom overview 2022-05-15                   173
```

```
#Creating a funciton to subset the dataframe between dates
mydatefunc <- function(x,y){Coronavirus_Data[Coronavirus_Data$date >= x & Coronavirus_Data$date <= y,]}

#Changing the class of the date column in the dataframe to 'Date'
Coronavirus_Data$date <- as.Date(Coronavirus_Data$date)

#Workflow to calculate dates
my_date <- max(Coronavirus_Data$date)
class(my_date)
```

```
## [1] "Date"
```

```
my_date_500 <- my_date - 500
my_date_300 <- my_date - 300

#Subsetting the dataframe for the last 500 days
Coronavirus_Data_500 <- mydatefunc(my_date_500,my_date)

#Calculating the date with the largest cases in the last 500 days
max_case <- max(Coronavirus_Data_500$newCasesByPublishDate, na.rm=T)
max_case_date <- subset(Coronavirus_Data, Coronavirus_Data$newCasesByPublishDate == max_case) %>%
  select(date)


#Finding the number of cases 300 days ago against recent
case_recent <- subset(Coronavirus_Data, Coronavirus_Data$date == my_date) %>%
  select(newCasesByPublishDate)

case_300 <- subset(Coronavirus_Data, Coronavirus_Data$date == my_date_300) %>%
  select(newCasesByPublishDate)

#Comparing Figures
if (case_recent>case_300) {
  'Case numbers have gone up'
```

```r
} else {
  'Case numbers have gone down'
}
```

```
## [1] "Case numbers have gone down"
```