July 16, 2015

1. Navigating in the linux Shell

$: Prompt, indicate the computer is ready for command

whoami

pwd: show current directory

cd: home directory

ls: list the content

ls –F F stands for flag, which adds a slash behind the directory. Used to distinguish files from directory

ls –a show all (including

ls . the same as ls (one dot indicates the current directory/level)

ls .. list one level up (../.. allows to jump two levels up)

ls –r display in reverse alphabetical order

ls \*[AB].txt

mkdir: make directory

nano draft.txt: pull up the text editor

touch draft.txt: create this file w/o pulling up the editor

rm draft.txt: remove file

rmdir: remove empty directory

rm –r: remove folder and everything underneath it

mv: move

mv file name 1 file name 2: rename

cp: cope file from one location to the other location

wc \*.pdb perform word count (line/word/character) of any pdb files

\* wild card

? extends to any single letter

wc –l only report lines

wc –l \*.pdb > lengths.txt redirect this output to put it in a file (lengths.txt)

cat lengths.txt printscreen of the content of the file

sort –n lengths.txt: sort the content

wc -l \*.pdb | sort -n |head -3 show the first three of the sorted content

wc -l \*.pdb | sort -n |tail -3 show the last three of the sorted content

< redirect eg. wc –l < my data.dat

>redirect overwrite into a file

>> double redirect append (add to the bottom of the last line in the file)

for loop:

> siqi-zhus-computer:creatures jsqzhu$ for **filename** in unicorn.dat basilisk.dat

> do

> head -3 **$filename** (display the first three lines in the said file, note the use of $)

> done

>siqi-zhus-computer:creatures jsqzhu$ for filename in \*.dat

> do

> echo $filename

(print to the screen, in this case print file name, or ### ${filename}### for readability)

> head -5 $filename | tail -3 (list the last three lines)

> done

bash goostats (filename output filename) call goostats (stats program)

>for datafile in \*[AB].txt; do echo $datafile; bash goostats $datafile stat-$datafile; done

find . –name \*.txt does not expand within the directory

find . –name ‘\*.txt’ expand within the directory

find . –type f

My\ Documents: back slash??

control r

control c: forget(forgo) what was typed

; can be used to replace ENTER

###: ignore what types after..

history

# comments in nano

man ls: display different command lines under ls

man grep:

Q: escape

Head: open the file..

2. RStudio (http://swcarpentry.github.io/r-novice-gapminder/)

script: highlight (or have cursor on the line) then apple+ENTER/run to display results in console

1 == 1: does 1 equal 1

1 !==1: does 1 not equal 1

* 1. == 0.3/3: floating point errors 🡪 should be all.equal(0.1, 0.3/3)

x <- 1/40: assign value

naming in R studio:

camelCase <- 5

dot.case

ls()

ls: source code of the function (without the bracket)

rm(list =ls()): remove everything

equal=: give argument to the function

vignette(): list of vignette of packages installed

maintained by CRAN: <https://cran.r-project.org/web/views/>

bioinformatics: http://bioconductor.org/

install.package(“”)

library()

#load installed package

assign variables: can be 5L(intergers), 2.2 (real numbers), pi, 1+0i(complex numbers), or “software carperntry”

x <- vector(“character”, length=10)

x <- c (1,2,3,4,5)

series <- 1:10

typeof() gives the different data types

is .logical(z): question is z logical?

Display:

Head(x) or head(x, n=2)

Tail(x)

Note: default n=6

str(x)

https://github.com/swcarpentry/r-novice-gapminder/blob/gh-pages/04-data-structures-part1.md

2D data:

set.seed(1)

x <- matrix (rnorm(18),ncol=6, nrow=3)

as.numeric(x): convert x to numeric

as character/logical etc…

factor: store logics (true false) as numerics

list(): allows different data type

eg. x <- list (1,”a”, TRUE, 1+3i)

df <- data.frame(id = c(“a”, “b, “c”, “d”), x = 1:4, y = 214:217)

typeof(df) is “list”

class(df) is “data. frame)

df <- rbind(df, list()):

my\_df <-data.frame(first\_name =, last name=)

str(my\_df)

class(my\_df$first\_name) will return factor we need character in order to rbind

my\_df$ <- as.character(my\_df$first\_name)

getwd(): get/set working directory

load up file: read.table(file="gapminder-FiveYearData.csv", header=TRUE, sep=",")

options()

sessionInfo()

[1] the first thing from the output

# for comment

?log to display help

??log to display fussy search help

5L(L stands for integer)

July 17, 2015

3. Git (https://en.wikipedia.org/wiki/Git\_(software))

Github: web interface for git; a distributed version-control system. Does not need internet; can work local

Configure git in terminal:

$ git config --global user.name "Siqi Zhu"

$ git config --global user.email "jsqzhu@gmail.com"

$ git config --global color.ui "auto"

$ git config --global core.editor "nano"

$ git config --global push.default current

pull from GitHub to local git repo “download”

$git clone <https://github.com/jsqzhu/sandbox-swc-2015-07-17.git>

cd to that directory (on desktop)

$git remote –v

#shows fetch/push from origin

#in order to establish the “upstream”

$git remote add upstream https://github.com/lwjohnst86/sandbox-swc-2015-07-17.git

#git does not run automatically in the bkgd

$git status

#to track the changes, show different status

$git init

#get git to start tracking this folder in the .git/ folder

$rm -rf git-playing/ (force remove)

$git add siqi-zhu.txt

$git commit

$git push #to upload

$git diff siqi-zhu.txt #shows the difference between different versions tracking changes

$git diff HEAD siqi-zhu.txt # shows the difference from the last commit

$git checkout HEAD~1#at staging area; useful in multi-team environment)

$git reset HEAD <file> #pull out of the staging area, tracked but not

$nano .gitignore #tells git to ignore certain file

.DS\_Store

ignore\*

4. More R

$curl http://blog.tomwright.ca/gapminder-FiveYearData.csv > gapminder-FiveYearData.csv

Summary(file)

Str(file)

> typeof(gapminder)

[1] "list"

> class(gapminder)

[1] "data.frame"

subsetting:

> populations <- gapminder$pop

> typeof(populations)

[1] "double"

> length(populations)

[1] 1704

populations[1] #sq bracket index into something

> populations[c(1,3,5)]

[1] 8425333 10267083 13079460

> square <- magic(5)

> square

[,1] [,2] [,3] [,4] [,5]

[1,] 9 2 25 18 11

[2,] 3 21 19 12 10

[3,] 22 20 13 6 4

[4,] 16 14 7 5 23

[5,] 15 8 1 24 17

> typeof(square)

[1] "integer"

> square[1]

[1] 9

> square[2,1]

[1] 3

> sum(square[2,])

>short <- populations[1:5]

>short

[1] 8425333 9240934 10267083 11537966 13079460

> short[c(TRUE,FALSE,TRUE,FALSE,TRUE)]

[1] 8425333 10267083 13079460

> index <-short >10000000

> short[index]

[1] 10267083 11537966 13079460

#find countries with populations>1e7

> population<-gapminder$pop

> bigpop<-population >1e7

> typeof(bigpop)

[1] "logical"

> gapminder[bigpop,]$country

#create a linear model

fit<-lm(pop~lifeExp,data=gapminder)

#lm=linear model, ~=given

> which(square[,1]>2 | square[,1]<16)

[1] 1 2 3 4 5

> which(square[,1]>2 & square[,1]<16)

[1] 1 2 5

# | or, & and

#which returns the index whereas square alone gives True/False

> gapminder[which(gapminder$year %in% c(1952,1957)),]$year

plot

ggplot(data = gapminder, aes(x = lifeExp, y = gdpPercap, color=continent)) +

geom\_point()+

scale\_y\_log10()+

geom\_smooth(method="lm",size=1.5)+

facet\_wrap(~continent)

#~means given

knit html/pdf

save as .md

# A heading

## A bigger heading

1. A numbered bullet
2. Another one
3. \* a bullet
4. \* A bullet

```{r}

x<-1:100

<http://stanford.edu/~dbroock/broockman_kalla_aronow_lg_irregularities.pdf>

Welcome to MoPad!

This pad text is synchronized as you type, so that everyone viewing this page sees the same text.  This allows you to collaborate seamlessly on documents!

Please be cognizant of whether you are using a public pad or private/team pad, and take appropriate precautions with data you post here

 Software Carpentry - SickKids

<https://etherpad.mozilla.org/YRZap32plJ>

Sample files:

questions 1:

<http://blog.tomwright.ca/files>

<http://swcarpentry.github.io/shell-novice/01-filedir.html>

<http://swcarpentry.github.io/shell-novice/02-create.html>

<http://swcarpentry.github.io/shell-novice/03-pipefilter.html>

R Stuff:

data: <https://github.com/resbaz/r-novice-gapminder-files>

github repo: <https://github.com/swcarpentry/r-novice-gapminder>

html files:

<http://swcarpentry.github.io/r-novice-gapminder/>

:

<http://swcarpentry.github.io/r-novice-gapminder/>

CRAN task viewer:

<https://cran.r-project.org/web/views/>

Bioconductor:

<http://www.bioconductor.org/packages/release/BiocViews.html#___Software>

ggplot manual:

<http://docs.ggplot2.org/current/>

r inferno:

<http://www.burns-stat.com/pages/Tutor/R_inferno.pdf>

Stackoverflow:

<http://stackoverflow.com/questions/tagged/r>

<http://stackoverflow.com/questions/5963269/how-to-make-a-great-r-reproducible-example>

Further resources:

advanced r - <http://adv-r.had.co.nz/>

r packages - <http://r-pkgs.had.co.nz/>

--------------------------------------------------------

R, day 2

Getting started, in your shell

* mkdir gapminder   # create new directory
* cd gapminder     # go to the directory
* git init    # initialize current directory in git
* curl <http://blog.tomwright.ca/gapminder-FiveYearData.csv> > gapminder-FiveYearData.csv  # goes to the URL downloads file to local directory
* go to RStudio, create new project using existing directory you just created (gapminder)

RStudio has several cheat sheets for ggplot, and markdown in particular. See:

<https://www.rstudio.com/resources/cheatsheets/>

<https://www.rstudio.com/wp-content/uploads/2015/05/ggplot2-cheatsheet.pdf>

<https://www.rstudio.com/wp-content/uploads/2015/02/rmarkdown-cheatsheet.pdf>

<https://www.rstudio.com/wp-content/uploads/2015/03/rmarkdown-reference.pdf>

the gpplot2 website at: <http://docs.ggplot2.org/current/> has good documentation about all of ggplot2s functions.

---------------------------------------------------------------------------------------

Git:

We are forking this repo - <https://github.com/lwjohnst86/sandbox-swc-2015-07-17>

fork it to your own! Then clone your version.

Here's the blackboard flow chart of git:

<http://imgur.com/SlOH85c>

Stackoverflow: <http://stackoverflow.com/questions/tagged/git>

**html lessons:** [**http://swcarpentry.github.io/git-novice/**](http://swcarpentry.github.io/git-novice/)

github: www.github.com

see the git repo of this lesson: <https://github.com/swcarpentry/git-novice>

LaCour paper controversy: <http://retractionwatch.com/2015/05/20/author-retracts-study-of-changing-minds-on-same-sex-marriage-after-colleague-admits-data-were-faked/>

Original criticism (written in R with knitr!): <http://stanford.edu/~dbroock/broockman_kalla_aronow_lg_irregularities.pdf>

Respinse to criticism (again written in R): <http://retractionwatch.com/wp-content/uploads/2015/05/LaCour_Response_05-29-2015.pdf>

Some Useful Git resources to explore:

Some tutorials <http://sixrevisions.com/resources/git-tutorials-beginners/>

Pro-Git Book: <https://progit.org/>

Git Documentation    <https://git-scm.com/doc>

Git How To   <http://githowto.com/>

Why version control?

* log writing activity
* remembers all changes and can go back to any saved point in composition
* makes open science easier (sharing and allowing other to verify your code)
* with intranets or internets, allows teams to work on same code and keeps track of who has changed what
* with use of a web remote, keeps back up of your work

git is the software; GitHub is the web platform

Configuring Git

git config --global user.name "yourfirstname yourlastname"

git config --global user.email "youemailaddress@somewhere"

git config --global color.ui "auto"

git config --global core.editor nano

git config --global push.default current   # tells git default directory to push from is the current

    directory

git remote -v  # to give you feedback on what you've set as "origin" and "upstream"

cat file.txt  # returns to screen the content of file.txt

git status   # returns status of your repository

git log   # returns log of activity on your repo

* git does not do "auto saves", you have to tell git when to record (take a snap shot) of your file

Git Work flow

* like R, Python, etc, you have to be in the directory containing your working files (your local repo) before you invoke git by typing "git" at the command prompt
* "git init"   # you tell git to start keeping track of current directory or folder
* creates a .git/ directory, which keeps track of your history, commits, etc; make sure you backup .git/ (for example, by using GitHub)
* You are working on your file in the "working area"
* You ADD ("git add")  your file to the STAGING AREA
* You have "added" your file but not "committed" it yet
* then you COMMIT ("git commit") your file from the staging area to your REPOSITORY
* you have now "committed" your file to the repository; git basically takes a snapshot of your file and stores it for you
* if you are using GitHub, you can then "push" your local repository to remote

Git setup for playing

* cd ~/Desktop
* mkdir git-playing
* cd git-playing
* git init
* ls -aF               # note that a .git directory has been created (where all the git stuff goes)
* nano play.txt    # add text "this is for playing around ..."
* git status          # see file is unknown
* git add play.txt # stage play.txt for committing
* git status          # see file is staged
* git commit        # commit the staged work
* git log               # see commit has been stored
* nano play.txt    # add some more text
* git diff HEAD    # compare directory to last commit

Moving files between your repository and the remote repository (over at GitHub) and GitHub Vocabulary

* "pushing" and "pulling"
* you "push" files from your local computer to the remote repository and you "pull" files from the remote to your local computer
* "origin"  or "upstream" is the local nickname of the remote repo
* "master" is the branch on of your local computer
* "clone"
* "fork"
* a  "pull request" is a message sent to owner or administrator of remote  repository saying that you have submitted a file (usually with changes)  and would like the owner to "pull" (accept) your file with its changes  and merge it with existing repo

Common Seven Verbs of Git

* git status
* git log
* git add
* git commit  (or git commit -m "your comments")
* git diff
* git push
* git pull

more git command examples:

* git diff file.txt      # returns changes to file.txt; only works on plain text files
* git pull upstream master  # pulls upstream master branch to local master branch
* git diff HEAD file.txt    # HEAD is the top of the history; returns most recent changes/difference
* git  diff HEAD~1 file.txt   # HEAD~1  means state of the file 1 before HEAD,  i.e. 2nd to last snapshot; you can go to a specific snapshot by  replacing "HEAD~1" with the first few characters of its hash number;  every commit has a unique hash number
* git checkout    # moves you to a different branch or moves you back in time
* git checkout HEAD~1   # moves your working directory to state of HEAD~1
* git checkout   # if no argument after "checkout", HEAD is assumed
* git reset HEAD file.txt   #  unstages your file.txt

Conflicts

* if  you pulled a file from the remote, modified the file, and pushed it  back to remote and someone else has also done the same thing, gitHub  will tell you there is a conflict and you have to resolve it between you  (and maybe the other person)
* if you get a conflict message,
* do a "git status" to see what's the issue
* do a "git diff" to see changes

Telling Git to ignore (do not track) files, directories

* create  a file named ".gitignore" and put in it names of files and directories  you do not wish Git to track (for example, confidential data files, scripts you are not ready to share yet, etc.)

R, Day2

-----------------------------------

mkdir gapminder

cd gapminder

git init

curl <http://blog.tomwright.ca/gapminder-FiveYearData.csv> > gapminder-FiveYearData.csv

Subsetting Data frame

three ways of doing this

* by name   (populations <- gapminder$pop)
* by indices  (e.g.  populations[1] , populations[1:4]
* logical indexing
* index <- short > 10000000   # returns logical vector (T/F values) for values in short
* short[index]   # use logical vector "index" to pull out values in vector "short" which are TRUE (> 10000000)

population <- gapminder$pop

bigpop <- population > 1e7

unique(gapminder[bigpop,]$country)

Installing a Package

* from RStudio, Tools, install package, give name of package
* from R command line: install.package("packagename")

Useful R commands:

* which(square[,1] >20)  #  returns row where column 1 entry is > 20
* "|"  means OR
* "&"  means AND

which(gapminder$year==1952 | gapminder$year==1957)  # gives you the indicies for where gapminder year is 1957 or 1952

# returns the actual observations

gapminder[which(gapminder$year==1952 | gapminder$year==1957),]$year

# another way of doing above action

gapminder[which(gapminder$year %in% c(1952, 1957)),]$year

calcMeanlifeExp<-function(data,year=NULL,country=NULL){

  if(!is.null(year)){

    data<-data[data$year %in% year,]

  }

  if(!is.null(country)){

    data<-data[data$country %in% country,]

  }

  lifeExp<-mean(data$lifeExp)

  return(lifeExp)

}

"

Kintr example

---

title: "test file"

author: "Tom Wright"

date: "17/07/2015"

output: pdf\_document

---

```{r echo=FALSE}

library(ggplot2)

```

# A heading

## A bigger heading

1. A numbered bullet

2. Another one

\* A bullet

\* A bullet

```{r}

gapminder<-read.csv('~/Desktop/gapminder/gapminder-FiveYearData.csv')

ncountries<-length(unique(gapminder$country))

```

There are `r ncountries` countries in this dataset

```{r}

ggplot(data=gapminder,

       aes(x=gdpPercap,fill=continent))+

  geom\_density(alpha=0.6)+

  facet\_wrap(~year)+

  scale\_x\_log10()

```