Software Tools Orientation

Research Computing and Data Management – Lab 1

Expectations and Grading

This lab assignment is worth two points of your participation grade. You will be required to turn in (through Canvas) evidence of your work (as described below). The assignment is due one hour after the end of the class session. Late lab submissions will lose one point.

Ask for help

Ask for help if you run into trouble. We will have some helpers available, so just raise your hand (or put a sticky note on your monitor) if you need help.

Before proceeding, please read the DISCLAIMER.

DISCLAIMER

While we encourage you to explore the capabilities and operation of your own personal computing devices, we caution you that the activities described in this course (and its laboratory exercises) do carry some risk to the computing device used. For this reason, we conduct the lab exercises in a classroom with computers provided for each student and we also provide access to a server which is managed on your behalf.

We are unable to provide repair services in the event that your own personal equipment malfunctions due to any of the activities suggested in the exercises or any other activity related to this course. If you choose to use your own personal equipment for the purposes of this course, please be aware that it will be your responsibility to address and correct any problems with your own personal equipment or data. We strongly encourage making backups of your important user documents and system files before attempting modifications to your computer such as the installation of new software or the configuration of system properties.

Please consult authoritative documentation and seek qualified assistance if you need help making backups or any other operation pertaining to the administration of your personal devices.

Software Installation - 1 pt

On the computer you are using in the lab (either the computer provided or your own laptop), make sure you have installed at least the more important software applications listed in Software Tools.

Since the Health Sciences Library (HSL) lab PCs do not have all of the applications we want, and will lose what we install upon reboot, we can use "Portable Apps" instead.

Portable Apps

These can be installed to your "U Drive" and will be available later.

- *Hint*: Store them in a folder such as, e.g., "U:\Apps". You will first need to create this "Apps" folder in your "U Drive" before proceeding.

You may also install them onto a portable device like a USB flash drive ("thumb drive") or external hard disk.

- Git
- ConEmu
- Notepad++
- jEdit

Apps you have already

The are already installed on HSL lab PCs:

- PuTTY
- FileZilla
- 7-Zip
- R
- RStudio
- Mozilla Firefox
- Notepad

Command-line utilities

These utilities will need to be installed to your U: drive (or USB media). They are very powerful command-line tools, commonly used in data management. OSX and Linux come with them, but Windows does not.

You can get many of the popular command-line tools for Windows by installing:

- Rtools - a collection of standard GNU utilities, including many GNU CoreUtils, diff, find, and grep, plus software compiling tools

So, please install Rtools to U:\Rtools. We will use the PATH "environment variable" to tell your computer how to find these when you try to execute them later.

Note: Many of these utilities also come with GitBash (with Git for Windows).

You might also find this command-line web browser handy to have around:

- cURL - For Windows, get the one labeled, "Win64 - MinGW64" and "binary", extract the archive, and place the binary executable file (`curl.exe`) in a folder in your PATH.

It also comes with GitBash, part of Git for Windows.

Apps to install as needed (but not today unless you have extra time)

These apps will need to be installed to C: drive, as needed. We will install these during ***future lab sessions***. As stated above, software installed to the C: (default) drive will be lost upon reboot of your HSL lab PC. This is because the HSL lab PCs have been configured to revert to a default configuration each time the computer is turned on. So, if you install the apps listed below to the C: drive, do not expect them to be there for you later. This would not normally be the case on a computer you own, or is managed in a different way, like a PC you have in a different lab setting. If you try to install them to the U: drive, the application files may be installed there, but important software library (i.e., "DLL files") and configuration (i.e., "registry") settings may still be written to C:.

- Google Chrome
- X2Go
- SQLite
  - Download the "command-line shell program" under **Precompiled Binaries for Windows**
- MySQL Workbench– Note: Windows users will also need:
  - Microsoft .NET Framework 4 Client Profile
  - and Visual C++ Redistributable for Visual Studio 2013
- Visual Studio Code
- York

What to turn in

You will be shown how to make a list of software (or enumerate the items in folders containing software) installed during the class session. Use this method to first make a list both ***before and after*** software installation and compare the two lists. Turn in the resulting comparison (a list). You may either paste in the list or upload it to Canvas as a plain-text file (with a .txt file suffix) attachment. (We will also ***show you*** how to see file suffixes in Windows, as well as how to make a "screen shot"– since these are both very handy to know.)

Command-Line Utilities - 1 pt

We will try out various command-line utilities from the "shell" (operating system command prompt within a "terminal"). The purpose of this exercise is to increase familiarity with this mode of computing and also to increase awareness of some of the interesting and useful things one can do from the command-line that may not be possible or as convenient to do from a graphical user interface (GUI).

One of the primary reasons to learn how to use these tools is to enable the automation of repetitive tasks which would otherwise be tedious and time-consuming using a mouse and GUI. Shell commands can be linked together into "scripts" (short programs) which can be run on thousands of files at once to quickly process data. Putting commands in scripts (i.e., "scripting") also improves reproducibility – the ability to repeat an operation at a later date (or by another person) and still get the same results.

Opening a terminal ("shell") session

In Windows, you can start DOS from the Start button by clicking the Start button and then typing CMD in the textbox and pressing the Enter key. You may also have a shortcut to the DOS Command Window, but with recent updates to Windows, it is harder to find these. Unix and Linux systems will often have a Terminal or Shell icon (or menu choice) that will launch a terminal, using defaulting to the Bash interpreter. If you connect to one of these machines with SSH, you will usually be presented with a Bash shell prompt once you login. On Windows systems with Git installed, you may also get a GitBash terminal by launching GitBash. You can do this is the same way you launched CMD (described above) or from a program icon. If you would like to have an icon for a program launcher stored in a handy place, you can make a shortcut. Lastly, if you installed a third-party terminal application like ConEmu, then you can launch that from the Program menu, etc., as with any other desktop application. By default, ConEmu will use the DOS command-line interpreter.

Commonly Used Command-Line Utilities

Examples of common utilitiesfound on Unix and Linux (i.e. "POSIX-compliant") systems are (with analogous utilities for Windows/DOS [in brackets]) are:

Folder (Directory) Navigation and Management

- cd, pwd [cd]
  - **c**hange **d**irectory (folder), **p**rint **w**orking **d**irectory (folder). (Note: "working" means the folder you are currently working "in".)
- mkdir, rmdir [mkdir (or) md, rmdir (or) rd]
  - **m**a**k**e (create), **r**e**m**ove a **dir**ectory (folder)

File Management

- ls, find [dir]
  - **l**i**s**t, **find** (search for) files and folders, etc. (using pattern matching with a special character syntax called "glob constructs")
- cp, mv, rm [copy, move, del, ren, rename]
  - **c**o**p**y, **m**o**v**e, **r**e**m**ove (delete), rename files (A move to a new name is a rename.)

File viewing and retrieval

- cat [type]
  - display (or combine, "con**cat**enate") the contents of text files (i.e., any screen output is displayed all at once)
- more, less [more]
  - "page" (display to screen with pausing) through (long) text files (with keyboard navigation)
- curl
  - fetch files from the web

Text manipulation

- echo [echo]
  - repeat text (printing it to the terminal screen or redirecting it to a file, etc.)
- grep [find]
  - find text (using pattern matching with a special character syntax called "glob constructs" or "regular expressions")
- diff [fc]
  - show **diff**erences between texts
- head, tail, cut
  - slice text in various ways: first n lines (**head**), last n lines (**tail**), **cut** (select) specific columns (fields, variables, features)
- sort, uniq [sort]
  - **sort** text: alpha-numerically, with or without duplicates (**uniq**ue)
- wc

  - **c**ount **w**ords, lines, and characters in text
  - Example: echo 'fooling' | wc -c # counts number of characters

- sed
  - **s**tream **ed**itor: make changes to text such as search-and-replace
  - Example: echo 'fooling' | sed 's/o/e/g' # changes all "o" to "e"

Environment

- export [set]
  - set (environment) variables. (In Bash, "export" makes them available to sub-processes. In DOS, "set" is used for assignment.)
- exit [exit]
  - end (close) the terminal session (and possibly write your command history to a file) – ***but save your terminal output first!***

We will also look at tab-completion, command-history navigation, I/O redirection, pipes, and job control, as much as we can (time permitting). Otherwise, we will finish covering those additional topics next week.

Exercise

Take a look at the `iris` data set at the UCI Machine Learning Repository. Read the "Data Set Information" and note how the "data differs from the data presented in Fishers article". How can we easily confirm this using the command-line tools we now have at our disposal?

First, using `bash`, make a folder, `iris`, with `mkdir` to store the data and then enter that folder with `cd`.

Second, using `curl`, download the two versions of the data set as CSV files. Use either the `-o` argument for the output file, or redirect the text to the output file using the `>` operator.

- "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"
  - Save as: "iris_data_1.csv"
- "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/bezdekIris.data"
  - Save as: "iris_data_2.csv"

How would we quickly confirm the differences mentioned at the UCI site?

R comes with the `iris` data set (built-in). Export it to a CSV file.

```
Rscript -e "write.table(format(iris, digits=2), 'iris_data_r.csv', sep=',',
    col.names=F, row.names=F, quote=F)"
```

(We wrapped this one-line command here, but enter it all on one line in Bash.)

Which of the two versions does R have? Since R's version lists the species in a different format, how can you easily ignore that difference and just look at the differences in numeric variables? (Hint: Use `sed` or `cut`.)

We can do a similar comparison in R using `sqldf`, which allow us to perform SQL queries on data frames. (SQL is the most common database query language.)

```
if (!require("sqldf")) { install.packages("sqldf"); require("sqldf") }
iris_data_r <- iris
url <- "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"
iris_data_1 <- read.csv(url, header = F)
names(iris_data_1) <- names(iris_data_r)
iris_data_1$Species <- gsub("Iris-", "", iris_data_1$Species)
iris_data_r$id <- row.names(iris_data_r)
iris_data_1$id <- row.names(iris_data_1)
cat("What's in R's iris data but not in UCI's iris data?", "\n")
sqldf('SELECT * FROM iris_data_r EXCEPT SELECT * FROM iris_data_1')
cat("What's in UCI's iris data but not in R's iris data?", "\n")
sqldf('SELECT * FROM iris_data_1 EXCEPT SELECT * FROM iris_data_r')
```

Save these commands as diff.R and run them using this command:

```
Rscript diff.R
```

Which method (`bash` versus `R`) is more work or more complicated? Is there an easier way to do this in either `bash` or in `R`?

If there is time, compare the files graphically in meld or Winmerge. Aside from ease of use, how else might these graphical tools make it easier to identify subtle differences? When would the command-line tools be preferable?

What to turn in

You will be expected to turn in your command history and output into Canvas, either as pasted text or as a plain-text file (with a .txt file suffix) attachment. ***You will be shown*** how to do this in class.