1 Introduction

I've been working as an analytics developer for nearly two years now, and wanted to share the most helpful things I've learned in my transition from academia.

First, there are a dizzying number of tools and techniques to learn. The general framework I want to cover is:

1. Setting up and navigating a development environment from the command line (I use OSX and Ubuntu)
2. Scripting with python and R,
3. Saving and sharing work with git and github, and
4. Storing and accessing data via databases and sql

This will be opinionated, in that I'll typically present what I do as the right way to do things. Obviously there are other ways of doing things.

2 Before You Start

We'll be moving mostly without a mouse. It is terrible at first, but pays dividends. If you are using Ubuntu you can get to a terminal with `ctrl+alt+T`. On OSX, use `cmd + space` to open spotlight, type in "terminal", then hit enter. I'll give some pointers to the most important commands to know in the terminal. Some important ones to start:

1. `$ ls` lists the files in the current directory
2. `$ cd directory` changes your directory to `directory` (if it exists).
3. The up/down keys scroll through recent commands
4. `tab` will autocomplete when it can. Sometimes you need to hit it twice and it will give you options
5. `ctrl+r` Gives you reverse search in the terminal, letting you type in a few letters. Hitting `ctrl+r` more scrolls backwards in these options, and `ctrl+g` cancel your search.

3 Setting up and navigating a development environment from the command line

If you can use an Ubuntu box, it will be much easier to set everything up, but it isn't terrible on OS X, either. In general, the better you can find your question on Stack Overflow, the better you'll do longterm, so check there early and often if you run into trouble.

We'll be using a few different terminals, and when you are meant to actively type something, I will try to copy the prompt: bash (my terminal) uses `$ `, Python uses `>>> ` and R uses `> `. Code with no prefix will be a `message printed back to you`.

Go to the appropriate section to set up your own environment for general purpose processing.

3.1 Ubuntu

We'll be using Ubuntu 12.04. On Ubuntu, you get the Advanced Packaging Tool, which we invoke with `$ apt-get`. We'll install a few other utilities off the bat:

Get git Start with `$ sudo apt-get install git`. Typing `$ git` should now output something reasonable. Bad news if you see `-bash: git: command not found`, and you'll have to do some research to fix this.

Get python Already built in – I'd just use theirs.

Get R We want 3.0.2, and for this we need to add their repository before we can `apt-get` R. Check out this page for good information.

3.2 OSX

Get Homebrew. Instead of `apt-get`, we'll use Homebrew. OS X ships with ruby, so you should be able to copy/paste the command at the Homebrew website with no problem.

Get git Now we can run `$ brew install git`. Typing `$ git` should now output something reasonable. Bad news if you see `-bash: git: command not found`, and you'll have to do some research to fix this.

Get python Python is already built into OS X (open your terminal and type `$ python` to start hacking away), but you want a different version of Python. Luckily, this is nice `$ brew install python`. This will install Python 2.7.6, which is what we'll be using. There are many discussions about Python 2 vs Python 3. We follow the advice of this one, and use Python2 while writing code that will be Python3 compatible. The main differences are that division is no longer integer division (in python2, `>>> 3/4 = 0`, in python3 `>>> 3/4 = 0.75` but `>>> 3//4 = 0`), more objects are generators, and the `print` function is invoked like a function.Python 3.4 was just released, but `$ brew install python3` will still install Python 3.3.3. Use at your own (minimal) risk.

Get R As usual, `$ brew install R` installs a good version of R (I am on 3.0.2).

3.3 Python Interpreter Hello World

Open your terminal, type `$ python` and then `>>> print("Hello, World!")`. That's a bit too easy. Let's instead run

```
$ python
Python 2.7.5 (v2.7.5:ab05e7dd2788, May 13 2013, 13:18:45)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> x = "Hello, World!"
>>> print(x)
Hello, World!
>>> print(x + x)
Hello, World!Hello, World!
>>> print(5 * x)
Hello, World!Hello, World!Hello, World!Hello, World!Hello, World!
>>> print(5 * (x + "\n"))
Hello, World!
Hello, World!
Hello, World!
Hello, World!
Hello, World!
```

Notice that you can assign strings to a variable, and manipulate strings using addition in multiplication (and it works in a sane fashion). The `\n` string is a newline. The other useful character to know for now is `\t`, which is a tab. To quit the console, `ctrl+d` or `>>> quit()`.

3.4 R Interpreter Hello World

Happily, we may again type `$ R`, and `> print("Hello, World!")`. We can't run exactly the same session as we did in Python, since R doesn't let us mix algebra and strings quite as nicely, but there are equivalents.

```
$ R

R version 3.0.2 (2013-09-25) -- "Frisbee Sailing"
Copyright (C) 2013 The R Foundation for Statistical Computing
Platform: x86_64-apple-darwin10.8.0 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> x <- "Hello, World!"
> print(x)
[1] "Hello, World!"
> print(paste(x,x))
[1] "Hello, World! Hello, World!"
> print(paste(rep(x, 5), collapse=""))
[1] "Hello, World!Hello, World!Hello, World!Hello, World!Hello, World!"
> cat(paste(rep(x, 5), collapse="\n"))
Hello, World!
Hello, World!
Hello, World!
Hello, World!
Hello, World!>
```

A few things to note about this:

1. We use the `paste` function to concatenate strings.
2. The `rep(x, 5)` created a vector with five copies of `x`. We could have done this by hand with `> c(x, x, x, x, x)`. More on R vectors later.
3. The `paste` function accepts a `collapse` argument, which tells it how to join a vector of strings.
4. In the last command, we used `cat` instead of `paste`. The `paste` function would print "\n", while the `cat` function interprets these as newlines.

5. The `cat` function doesn't automatically put a newline after it does its thing, which is why the interpreter starts typing after the last "Hello, World!"

4 Tools for using R

R has some fantastic support for getting up and working quickly and easily. A few ways to help get hacking:

Defining variables. R uses the symbol `<-` instead of `=`. Using `=` will still work (see this article if you're interested), but it is typically reserved for setting function arguments.

Using Packages You can import R code in a number of ways. I just use `> require(ggplot2)` to import, for example, ggplot2. You could also use `> library(ggplot2)`, with almost identical results. If I am importing a local file, use `> source('../relative/path/to/myfile.R')`. Note that with this, you can run code, assign values to variables, and define functions. A good workflow is to have an interactive console open, figure out how to make R do what you want to do, copy that over to a file, and `source` the file.

Installing Packages. A great strength of R is that it has some cutting edge packages available. These are uploaded to CRAN, the "Comprehensive R Archive Network", which is "a network of ftp and web servers around the world that store identical, up-to-date, versions of code and documentation for R". The majority of packages you'll want are installed via `> install.packages("ggplot2")`. You'll be asked to choose a mirror, and typically the best will be the cloud mirror that RStudio maintains on an Amazon EC2 instance – number 0. It syncs with the central CRAN mirror in Austria once a day, and is typically beats any other mirror for speed (in Austin, the cloud mirror is faster than using the mirror in Dallas).

Occasionally, you'll want something really cutting edge that you find on github. There's a nice package from Hadley Wickham that makes this easy:

```
> require(devtools)
Loading required package: devtools
```

```
> install_github('ggplot2')
Installing github repo ggplot2/master from hadley
Downloading ggplot2.zip from https://github.com/h...
Installing package from /var/folders/k1/3qh0v1017...
arguments 'minimized' and 'invisible' are for Win...
Installing ggplot2
'/Library/Frameworks/R.framework/Resources/bin/R'...
  '/private/var/folders/k1/3qh0v1017ql8p4xm7hqzv_...
  --library='/Users/colinc/Library/R/3.0/library'...

* installing *source* package 'ggplot2' ...
** R
** data
*** moving datasets to lazyload DB
** inst
** tests
** preparing package for lazy loading
** help
*** installing help indices
** building package indices
** testing if installed package can be loaded
* DONE (ggplot2)
> require(ggplot2)
Loading required package: ggplot2
> sessionInfo()
R version 3.0.2 (2013-09-25)
Platform: x86_64-apple-darwin10.8.0 (64-bit)

locale:
[1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8

attached base packages:
[1] stats     graphics  grDevices utils
[5] datasets  methods   base

other attached packages:
[1] ggplot2_0.9.3.1.99 devtools_1.4.1

loaded via a namespace (and not attached):
 [1] colorspace_1.2-4   dichromat_2.0-0
 [3] digest_0.6.4       evaluate_0.5.1
 [5] grid_3.0.2         gtable_0.1.2
 [7] httr_0.2           labeling_0.2
 [9] MASS_7.3-29        memoise_0.1
[11] munsell_0.4.2      parallel_3.0.2
[13] plyr_1.8           proto_0.3-10
[15] RColorBrewer_1.0-5 RCurl_1.95-4.1
[17] reshape2_1.2.2     scales_0.2.3
[19] stringr_0.6.2      tools_3.0.2
[21] whisker_0.3-2
>
```

Note the use of `> sessionInfo()` at the end to view all the attached packages and their versions.

Getting help. It is notoriously difficult to google for help with R, given

its name (though, really, who hasn't come up with unexpected results while searching for help with LaTeX, Python, or Ruby? I've even heard a story of a trip to manpages.com in search of bash help.) The built-in R help pages are accessed via `?`, and are actually quite helpful.

Note that `?` will only search packages which are currently loaded. Try the following session:

```
> ?geom_point
No documentation for 'geom_point' in specified packages and libraries:
you could try '??geom_point'
> require(ggplot2)
Loading required package: ggplot2
> ?geom_point
```

The suggestion to use `??` is also a good one: when I type it in, I see

```
Help files with alias or concept or title
matching 'geom_point' using regular
expression matching:


ggplot2::geom_point
                Points, as for a scatterplot
ggplot2::geom_pointrange
                An interval represented by a
                vertical line, with a point
                in the middle.


Type '?PKG::FOO' to inspect entries
'PKG::FOO', or 'TYPE?PKG::FOO' for entries
like 'PKG::FOO-TYPE'.




(END)
```

This command has searched through all *installed*, not necessarily loaded, packages. If you see a sweet function on a blog post and want to find which package it is in, you need to either go to google, RSeek, crantastic, or use the `sos` package:

```
> install.packages('sos')
Installing package into '/Users/colinc/Library/R/3.0/library'
(as 'lib' is unspecified)
--- Please select a CRAN mirror for use in this session ---
trying URL 'http://cran.rstudio.com/bin/macosx/contrib/3.0/sos_1.3-8.tgz'
Content type 'application/x-gzip' length 213172 bytes (208 Kb)
opened URL
==================================================
downloaded 208 Kb
```

```
The downloaded binary packages are in
  /var/folders/k1/3qh0v1017ql8p4xm7hqzv_3m84fy03/T//Rtmp3Dpn1e/downloaded_packages
> require(sos)
Loading required package: sos
Loading required package: brew

Attaching package:    s o s

The following object is masked from   package:utils  :

    ?

> findFn('geom_point')
found 197 matches;  retrieving 10 pages
2 3 4 5 6 7 8 9 10

Downloaded 175 links in 50 packages.
>
```

This creates a local website with links to all the packages that you might find `geom_point` in.