EPRDCTN in the command line

Introduction

This series of posts intends to walk you through some basic concepts, activities and shell commands to take the fear and pain away from working with a text-based interface that links directly to the Operating System.

In more formal terms:

> A command-line interface is a means of interacting with a computer program where the user (or client) issues commands to the program in the form of successive lines of text (command lines). A program which handles the interface is called a command language interpreter or shell.
>
> Wikipedia

So what does it mean?

It’s like an old style terminal where you enter commands that make the computer do something.

All Operating Systems have a CLI. Yes, even Windows and MacOS.

screenshot of a Bash shell in the GNOME windows manager for Linux

Figure 1: Screenshot of a Bash shell in the GNOME windows manager for Linux

screenshot of Windows Powershell as it works in Windows Vista

Figure 2: Screenshot of Windows Powershell as it works in Windows Vista

This is important because sooner or later you will find tools that will only work from a command line interface. We'll explore some of\` \`\`\`\`\`\`\`\`\`\`\`\`\`\`\`\`\`\`\`\`\`\`\`\`\` these tools (Node, Daisy Ace) in later sections but it's important make this clear.

What command line tools will we use?

In Windows the better tools are PowerShell a souped up terminal shell with additional scripting capabilities, and Windows Subsystem for Linux (WSL) a way to run Linux native applications from Windows. It uses a Ubuntu Linux image for Windows, not a modified version of Linux to run on Windows but a full version of Ubuntu Linux that will work together with Windows.

As far as terminals are concerned we'll use the iTerm2 for the Mac and a standard Bash shell for WSL.

Before we get started

Before we jump into further installations and customizations we need to do a few things that are dependent on the Operating System we're using.

Mac Users: Install XCode Command Line Tools

Before we install Homebrew we need to install Xcode command line tools. These are part of the full Xcode download but I'd rather save you the 5GB+ download so we’ll go the slim (but with more steps) route instead.

1. Go to the Apple Developer’s site
2. Click on the account link on the right side of the top navigation bar. You can use the same account that you use of iTunes or any Apple property.
    1. If prompted verify your account. This mostly happens when logging in from a new location or with a new computer
3. Click on Download Tools
4. Scroll down the screen and click on **See more downloads**
5. On the search box (to the left of the list of items to download) enter **Command Line Tools**. This will reduce the number of entries
6. Download the version that matches your MacOS version
7. Install the package.

The version I downloaded was 173MB. I’m OK with the extra work :)

Command Line Tools For Xcode download screen

Figure 3: Command Line Tools For Xcode download screen

Windows Users: Make sure WSL and the Ubuntu Image are installed

Before we move forward with WSL and Linux on Windows we need to make sure we have the right version of WSL installed and that we downloaded Ubuntu from the Microsoft Store.

> These instructions assume you’re using the latest veersion of Windows 10.

1. Install the latest version of PowerShell
    1. Download the MSI package from our GitHub releases page. The MSI file looks like this - PowerShell-6.0.0.`<buildversion>.<os-arch>`.msi
    2. Once downloaded, double-click the installer and follow the prompts.

        There is a shortcut placed in the Start Menu upon installation.
    3. By default the package is installed to `$env:ProgramFiles\PowerShell\`
    4. You can launch PowerShell via the Start Menu or `$env:ProgramFiles\PowerShell\pwsh.exe`
2. Install WSL from PowerShell as Administrator
    1. Type powershell in the Cortana search box
    2. Right click on Windows PowerShell on the results and select Run as administrator
    3. The UAC prompt will ask you for your consent. Click Yes, and the elevated PowerShell prompt will open
3. In the PowerShell window you just opened type: `Enable-WindowsOptionalFeature -Online -FeatureName Microsoft-Windows-Subsystem-Linux`
    1. Reboot the system when prompted
4. Install your Linux Distribution
    1. Open the Microsoft Store and choose your distribution. We'll be working with Ubuntu; other distributions are presented for reference.
        1. Ubuntu
        2. OpenSUSE
        3. SLES
        4. Kali Linux
        5. Debian GNU/Linux
5. Select "Get"
6. Once the download has completed, select "Launch".
    1. This will open a console window. Wait for installation to complete then you will be prompted to create your LINUX user account
7. Create your LINUX username and password. This user account has ***no relationship*** to your Windows username and password and hence can be different

Installing a terminal on the Mac

Even though there is a terminal bundled with MacOS (hidden inside applications -> utilities) I like iTerm 2 as a more feature complete replacement for the terminal that comes with MacOS. You can download it from the iTerm 2 download site and, please, make sure you download the stable release.

Package Managers

Most Operating systems have ways to automate software installation, upgrade, management and configuration of the software on your computer with package managers.

Package managers are designed to eliminate the need for manual installs and updates. This can be particularly useful for Linux and other Unix-like systems, typically consisting of hundreds or even tens of thousands of distinct software packages.[2]

We'll look at Homebrew and apt-get, their requirements and ecosystems, along with some basic commands to get you started.

Homebrew and Cask

Homebrew allows you access to a large ecosystem of Unix Software on your Mac. It is a Ruby application which is one of the reasons why we installed the Xcode command line tools; they include Ruby.

To install Homebrew paste the following command on your terminal. This will download, install and configure Homebrew.

```
/usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/
```

Now that we have installed Homebrew we'll use it to install, upgrade and remove (uninstall) a package. Even though I've chosen to install a single package. The same would be applicable to single package and multiple package installs.

This and the following sections will install the `akamai` package.

Installing the package

Installing packages is simple. The command is:

```
brew install akamai
```

The command will also install any dependencies needed for the package to run.

Akamai has no dependencies.

Screenshot showing the install process for a homebrew package

Figure 4: Screenshot showing the install process for a homebrew package

Updating/Upgrading the package

We should periodically update our packages to make sure we're using the latest version and capture any/all packages. The upgrade process gives us two options, one is to upgrade individual packages with an example like the one below:

```
brew upgrade akamai
```

If the package you're upgrading individually is already up to date, Homebrew will present you with this 'error' message. It's not an error at all, just Homebrew's way of telling you it's not needed.

alt-text

Figure 5: alt-text

The other option is to upgrade all installed packages at the same time by just using

```
brew upgrade
```

Homebrew upgrade process for all packages

Figure 6: Homebrew upgrade process for all packages

Uninstalling the package

When we’re done, we can uninstall the package to free up space in the hard drive (always a concern for me). The command is

```
brew uninstall akamai
```

Cleaning up after your installed packages

OK, I’ll admit it… I have packages in my Homebrew installation that I haven’t used in ages but, sooner or later my hard drive will complain and force me to cleanup old stuff. With homebrew this is simple, the command is:

```
brew cleanup
```

This will go through all installed packages and remove old versions. It will also report when it skips versions because the latest one is not installed and how much hard drive space it gave you back

Homebrew cleanup showing a listing of removed packages and how much disk space was saved in the process.

Figure 7: Homebrew cleanup showing a listing of removed packages

and
how
much
disk
space
was
saved in
the
process.

There are more commands to use when troubleshooting and building recipes for Homebrew but the ones we've covered are the basic ones you'll use most often.

Cask: Like Homebrew but for applications

I don't particularly care for the way you have to install some software for MacOS. You download the file, open it then drag the application to the applications folder in your Mac (usually aliased in the folder created by the installer) and only then you can actually use the program.

The creators of Homebrew put out Cask, a command line software installer. The installation is simple, paste the following command on your terminal:

```
brew tap caskroom/cask
```

Then you can use Cask to install software on your system. For example, to install Java, run the following command.

```
cask install java
```

Cask will accept ULAs and other legal agreements for you. If these type of agreements are important **do not use Cask** and install software the old fashioned way.

Apt-get and apt-cache for Windows (?)

Linux is built around the concept of packages. Everything in a Linux distribution from the kernel, the core of the Operating System, and every application is built as a package. Ubuntu uses APT as the package manager for the distribution.

There are two commands under the `apt-get` umbrella: apt-get itself and apt-cache. apt-get is for installing, upgrading and cleaning packages while apt-cache is used for finding new packages. We'll look at the basic uses for both these commands in the next sections.

In the following sections `ack` is the name of the package we'll be working with, not part of the commands.

Update package database with apt-get

apt-get works on a database of available packages. You must update the database before updating the system, otherwise apt-get won't know if there are newer packages available. This is the first command you need to run in any Linux system after a fresh install.

Updating the package database requires super user privileges so you'll need to use sudo.

```
sudo apt-get update
```

Upgrade installed packages with apt-get

Once you have updated the package database, you can upgrade the installed packages. Using `apt-get upgrade` will update all packages in the system for which an update is available. There is a way to work with individual packages that we'll discuss when installing new packages

```
# Upgrades all packages for which an update is available
sudo apt-get upgrade
```

You can also combine the update and upgrade command into a single command that looks like this:

```
# Combines both update and upgrade command
sudo apt-get update && sudo apt-get upgrade -y
```

The logical and will make sure that both commands run and will fail if either do. It is the same as running the commands individually.

Installing individual packages

After you update your system there is not much need to update it again. However you may want to install new packages or update individual packages. The `install` command will do either.

```
sudo apt-get install ack
```

If the package is not installed, the command will install it, along with its dependencies and make the command available to you.

If you've already installed the package, either during an upgrade or manual install, the command will compare the installed version with the one you want to install, if the existing version is the same or newer the installer will skip and exit. If the version being installed is newer then the installer will execute the upgrade.

Uninstalling individual packages

There are a few times when a package breaks stuff somewhere else or you no longer need the functionality the package provides. In this case you can do two things.

You can use the `remove` command to only remove the binaries, the applications themselves, and leave configuration and other auxiliary files in place. This will make it asier to keep your configuration without having to write it down.

```
# ONLY REMOVES BINARIES
sudo apt-get remove ack
```

The next, and more extreme, option is to use the `purge` command. This will get rid of all portions of the package, beyond what the `remove` command will do. Use sparingly if at all.

```
# REMOVES EVERYTHING, INCLUDING CONFIGURATION FILES
sudo apt-get purge ack
```

Cleaning up after yourself

Just like with Homebrew, apt-get will keep older versions of installed packages. Sooner or later your system will complain about being low on resources and will require you to clean up the system

The first option is to run the `clean` command. This will clean your local system of all downloaded package files.

```
sudo apt-get clean
```

The second options is the less extreme `autoclean` command. This will only removes those retrieved package files that have a newer version now and they won't be used anymore.

```
sudo apt-get autoclean
```

apt-search to find packages

There are times when you're looking for someting but are not sure exactly what. This is where the `apt-cache search` command comes in, if you enter a search term it'll find all related packages.

```
apt-cache search <search term>
```

If you know the exact package name you can use `apt-cache pkgnames` command that will return all package names that match your search criteria. The number of returned items will be smaller than the search return.

```
apt-cache pkgnames <search_term>
```

Node

Node.js (or just Node) is a cross platform Javascript interpreter built on the [V8](V8), the same Javascript interpreter that powers Google Chrome. Initially Node was created to run Javascript on the server but it has also been used to create a lot of tools for use on personal computers. This is the side of Node that we'll concentrate in.

In this section we'll look at the following aspects of Node:

- Installing Node with NVM
    - Why I chose this method
- Installing, removing and updating packages
    - Global install
- Some examples of what you can do with Node

Installing Node with NVM

NVM is a set of shell scripts that allow you to download, configure and use multiple versions of Node without conflict. It installs and configure the Node binaries to run from your home directory, avoding potential permission issues.

To install NVM open your terminal (or WLS though PowerShell) and paste the following command:

```
curl -o- https://raw.githubusercontent.com/creationix/nvm/v0.33.8/install
```

This will download NVM, configure your directory and set permissions appropriately.

The next step is to actually install Node. We'll install a LTS (Long Term Support Version) 8. I'm installing a LTS version to make sure we have the best chance of our packages being supported. The latest versionn (9.x) may have changes that will not work with our software.

To install the latest release of Node 8.x run the following command:

```
nvm install 8
```

This will install the latest version that matches the major number you chose to install. As of this writing the latest version is `8.11.1`. To activate the version you just installed type:

```
# Replace 8.11.1 with the version you installed
```

```
nvm alias default 8.11.1
# Use the default version you just configured
nvm use default
```

Note: Installing newer versions of Node with NVM will not delete older versions. For example, you can install Node 8.12 and still switch back and forth between versions using node `use` and the version you want to use for a specific project.

If you've already installed packages for a version of Node you can migrate the packages to the new installation. To do so run the following command (using a hypothetical version 8.12)

```
nvm install 8.12 --reinstall-packages-from=8.11
```

Installing packages from an older version requires you to specify the version you're installing from. It is not enough to say version 8... you're installing the latest release from that version, right?

Installing Node Modules

Installing Node also installs NPM, the Node Package Manager. This is the name of the tool and the ecosystem and repository that has evolved around it. *We'll use NPM to refer to the command line tool we use to install modules*.

Node packages are called modules and can be installed in one of two ways:

- ***globally*** meaning that they are available everywhere and usually provide a command line tool for you to work with
- ***project-based*** meaning that they are only available for the package they are installed under

Most of the tools that we'll use in an #eprdctn workflow fall in to the first category so I'll concentrate on global installation and package management. If you think this is a mistake open an issue on Github or contact me on Twitter (@elrond25).

For these examples we'll use the Ace accessibility checker from the Daisy Consortium

```
npm install -g @daisy/ace
```

This will install the module and produce a binary tool for you to run: `ace`. To check the version of the tool you're using run the following command:

```
ace --version
```

This installation introduces another concept worth paying attention to: scoped packages as a way to keep your group related packages together/ It also affect a few things about the way npm treats the package.

Removing Modules

There are times when we need to remove modules. The command is:

```
npm uninstall -g @daisy/ace
```

This will cleanup the command executables and all related configuration.

Upgrading Node Modules

Nodes lets you update all packages you've installed globally that have a version different from latest. The command is:

```
npm update -g
```

Unlike installing and uninstalling modules, you can run the update command without a specific file name and it'll work with all packages you've installed globally.

Java: JRE vs JDK, oh my!

We still use Java for some applications like Epubcheck. The advantage of working with Java applications is that you run the same application in all platforms (Windows, Mac and Linux) without customizing for each operating system.

Different versions of Java: JDK and JRE

You may hear the words Java, JDK, JRE thrown out when people talk about Java software. Let's try to cllear some of the confusion.

- **Java** is the language
- **JDK** is the Java Development Kit. It contains all the tools that you need to compile and run applications written in the Java language
- **JRE** is the Java Runtime Environment. It allows you to run Java applications but **not** compile or build them

To run applications either the JRE or JDKK will work. For simplicity sake, we'll install the JDK in the examples below.

Installing and Managing Java on the Mac: Use Cask

As discussed earlier in the Package Management Homebrew has a software management script called Cask. We'll expand on that section to cover installing, uninstalling and removing older versions of software using Java as an example.

Cask will accept ULAs and other legal agreements for you. If these type of agreements are important **do not use Cask** and install software the old fashioned way.

Installing Cask

Install Cask with the following Homebrew command.

```
brew tap caskroom/cask
```

This will make the `cask` command available. To verify the installation run:

```
cask --version
```

If the command is successful it should print something like the content below to the screen.

```
Homebrew-Cask 1.5.13
caskroom/homebrew-cask (git revision 63231; last commit 2018-03-30)
```

Installing software

To install Java use the following command.

```
cask install java
```

This will download the software, install it (accepting any required EULA or license) and make the software available to you.

Updating Software

Java is notorius for quick updates and security fixes. It appears that, along with Adobe Acrobat, Java is a favorite target of hackers and malware writers.

You should get into the habit of periodically updating your version of Java and testing your applications against the new version.

The command:

```
cask upgrade Java
```

Reinstalling Software

There are times when things break. Configuration files may get corrupted or you may accidentally delete something on a package that you didn’t mean to.

Rather than uninstall and reinstall cask gives you the option of reinstall. This command reinstalls the application and leaves it as if you just installed it for the first time.

```
cask reinstall java
```

Installing Java on Windows

Rather than automate the installation on Windows, I think the best way to go in Windows is to manually install Java as a Windows application and run that version from either Windows or Linux by tking advantage of the interoperabillity WSL provides (subject of a later section).

To install Java on Windows:

- Go to the Java Manual download page
- Click on Windows Offline
    - The File Download dialog box appears prompting you to run or save the download file
    - Click Save to download the file to a known location on your local system, for example, your desktop
- Close all applications including the browser
- Double-click on the saved file to start the installation process
- The installation process starts. Click the Install button to accept the license terms and to continue with the installation
- Oracle offer various products to install alongside Java. You are not required to install any of these partner products and their instllation (or lack thereof) should not affect Java at all
- A few dialogs confirm the last steps of the installation process
    - Click Close on the last dialog. This will complete Java installation process.

Basic shell commands, pipes, globes and others

Since we’re working on the command line we also need to learn about what makes the Unix/Linux command line work.

Globes

There are times when we need to match commands against one or more files. It can be as simple as listing only files of a certain type or having the command run only on some files, not others.

Unix-like shells (like those on MacOS and WSL) provide globs (global or pattern matching) to help you with the seaarchhes. I could go on for pages and pages about globs, but rather than do that I will give you some examples and let you explore further on your own.

The two most often used patterns are * and ?.

*

Matches any string, including the null string.

?

Matches any single character.

The following command will search the current directory for files that end with `.xhtml`. This can be useful to get a listing of all files of a given type in a directory.

```
# Search for all XHTML files in the current directory
# We define XHTML files as any file that ends with .xhtml
ls -al *.xhtml
```

The ? pattern matcher matches a single character. This is useful when you have a sequential list of files that are different by a single character. The example below would match all files that start with the word index, a single character and then end with .html. It Would match index1.html but not index10.html or index.html

```
# Search for all files that start with the word index,
# a single character and then end with .html
# Would match index1.html
# but not index10.html or index.html
ls -al index?.html
```

Newer version of Bash will let you recursively match using the ** pattern. This is not guaranteed to work everywhere so it’s offered here for you to try it.

```
# sets the recursive match option
shopt -s globstar
# Recursive search the current and all children directories
# for python files (anything that ends with .py)
ls **/*.py
```

For more information check this wiki page. It contains much more indepth and detailed information about glob patterns.

Pipes

A pipe pipeline is a sequence of one or more commands separated by the control operators `|`. The idea is that we run the first command then process the output of the next command (left to right) until we've ran all the commands on the pipe.

The following command will create a listing of all the files in a directory and look for the word index in the result

```
ls -al | grep index
```

Redirection

Another type of pipe is used to redirect the output of a file into another, usually a text file or some kind. The most typical example is piping the output of a command, in this case `ls -al` to a text file.

```
ls -al > content.txt
```

Creating aliases: What, Why and How?

There are times when typing the same command, particularly if it's a long command or one with many parameters, can be tedious and cause more errors than we care for. The bash shell provides ways to create shortcuts in the way of aliases.

You can create aliases for the current session using the alias command in the current shell. For example, if you paste the following command on your terminal:

```
alias ll="ls -lhA"ls -lhA"
```

It will create the `ll` command which will list all fikes and directories in long format skipping the current and parent directories.

The problem with this approach is that the command will only last as your logged in to the shell where you entered it. When you log in the commands are gone.

There is a way to create aliases that will persist through different shells. This works in Mac and, somewhat, in Windows though WSL.

First, make sure that the Nano package is installed in your Mac via Homebrew

```
brew install nano
```

The following steps are the same for Mac and Windows

```
# Makes sure you're in your home directory
cd
# Opens or creates the file .bash_profile
nano .aliases
```

Nano editor working on aliases file

Figure 8: Nano editor working on aliases file

When in the editor opens the file you will see an empty file or the result of your prior work.

Enter the new alias that you want to keep at the bottom of the file. If you’re entereing multiple aliases the order in which you do is not important.

Bash will read the entire file before starting and will flag any syntax errors.

There are two ways to get Bash to pick up the changes to your `.aliases` files. One is to open a new shell. The second one is to source the aliases files. The command is:

```
source .aliases
```

This command will force Bash to read the `.aliases` file and make all the aliases you created available to the shell.

The last thing to do is to make sure `.aliases` is loaded when we login. Add the following to the `.bash_profile` file

```
source .aliases
```

The path

In the Unix world, the path is the list of directories the shell follows when trying to find your applications. If the location of the application (expressed as a Unix path) is not on your path, then you will get a `command not found` error.

As with aliases you can set them one of two ways:

For the current shell (and only for the current shell) you can export the path. For example, if I wanted to add the program `foo` to my sell path for the current shell, I can do:

```
export PATH="$PATH:/usr"$PATH:/usr/local/bin/foo"
```

The export command is used to export a variable or function to the environment of all the child processes running in the current shell.

`$PATH` represent the current value of the path variable. We add it first to make sure we can still run all the system's built-in commands. We use a colon, `:` as the separator.

The last item is the path to the command we want to use.

Doing this every single time we want to run the command is tedius. Like we did with the aliases we can do the same thing with path commands by editing the `.bashrc` file.

The example below shows multiple instances of adding to the path. These are all additive; they will append to the path, not overwrite it.

The example below shows how to add multiple programs to the path one at a time using multiple export commands where there is more than one path (like between $PATH and the first item) they are separated by colon (`:`). This make it easier to change individual components if needed.

```
# Path to depot tool to work with Google code and other stuff
export PATH="$PATH:/Users/carlos/code/depot_tools"
"$PATH:/Users/carlos/code/depot_tools"# Shaka Packager
export PATHger"
# Homebrew
export PATH="/usr/local/sbi"
# Homebrew
export PATH="# Homebrew
export PATHTH"
# AV1 from source
export PATH="$PATH:/Users/carlos/code/aom-depl"
# AV1 from source
export PATH="# AV1 from source
export PATH="$PATH:/Users/carlos/code/aom-deploy/aom-build:/Users/carlos/
```

To add the export commands to your `.bashrc` file use the following commands using the Nano editor.

```
# to make sure we're in the home directory
cd
# open the file with the nano editor
nano .bashrc
```

The syntax of the command is the same as if you were adding it to the individual shell.

One final thing is to make sure `.bashrc` is loaded when we login. Add the following to the `.bash_profile` file

```
source .bashrc
```

WSL Interoperability

One important thing that Windows users should know. There is a reason why we went through all the work of installing PowerShell and WSL: Interoperability.

WSL and Powershell were designed to work together and are continuosly improved in the interoperability front. This can take one the following ways:

You can run Linux commands from Windows prepending the `wsl` command to the command you want to run. The example below runs Ace installed via NPM on Linux.

If this meets your needs, then you don't need to install Node on Windows. Running `wsl` node will take care of it when working on PowerShell and node will work when running on WSL.

```
wsl ace --help
```

You can mix commands too. The example below shows how to run a Windows command (`dir`) and pipe it to a Linux command with `wsl` (`wsl grep foo`).

```
dir | wsl grep foo
```

You can also do it the other way around and use Windows applications from the Linux shell. For example if you wanted to use Java to run Epubcheck from within a Linux shell you could run:

```
java.exe -jar epubcheck.jar
```

The `.exe` suffix to the application is important. It is what tells WSL that it's a Windows application; it also means that you don't have to install applications in both places, installing them in Windows should be enough.

Note that these commands don not work with aliases. As far as I know I can't create an alias in WSL’s Bash shell and run it using `wsl` + alias.

Links and Resources

- WSL
    - WSL Documentation
    - WSL interoperability with Windows
- Homebrew
    - Homebrew Website
- APT-GET Guide
    - Using apt-get Commands In Linux
- Lynda.com Courses
    - Workflow Tools for Web Developers
    - Unix for MacOS X Users