# Useful command line utilities specific to OS X:

systemsetup –

configuration tool for certain machine settings in System Preferences. Requires to be run with elevation (sudo or as root).

systemsetup -help

systemsetup -get|setremotelogin [-f (force)] [on|off] Enable/disable SSH daemon

systemsetup -get|setsleep [int in minutes]

-setsleep will configure display & hard drive. Computer needs to be specified individually.

systemsetup -setcomputersleep [int in minutes]

systemsetup -setrestartfreeze [on|off] restart after kernel panic.

systemsetup -setcomputername [value]

networksetup –

configuration tool for network settings in System Preferences. Requires elevation for system-wide settings only, but none for user scope settings.

networksetup –listallnetworkservices will show system names for configured network services

networksetup –setv6off [network service name] disable IP6 on specified interface. Recommended for AirOrangeX configuration.

networksetup –setnetworkserviceenabled [network service name] [on|off] enable or disable a specific network service

Other command line utils that I have not covered, but should be on your list of tools (‘man [command]’ and google are your friends here):

* defaults
* plutil
* /usr/libexec/PlistBuddy
* scutil (specifically scutil --nc for VPN connection control)

# Profiles

Tools used:

McxToProfile.py (<http://github.com/timsutton/mcxToProfile)>

TextWrangler (<http://barebones.com> - also through your local Munki repo)

“profile” command.

Profiles can be used for system and application preference deployment. This is very useful for bringing a machine into a desired state at the time of deployment, as well as compelling specific configurations.

Profiles may be applied a single time for a user, or every time the machine is used – e.g. set up a dock with custom applications and either allow the end user to modify it moving forward, or force the dock to remain in the specific state (in a lab environment).

Data is stored in XML format – may be plain text, binary, or signed/encrypted. TextWrangler will handle both Text and Binary formats.

Naming convention follows a reverse domain name construct to prevent filename collisions.

Note: Never edit preference files directly! We are opening them in TextWrangler for READ ONLY purposes.

Demo examples:

1. Locate “com.apple.finder.plist” in ~/Library/Preferences/ and open in TextWrangler
2. Copy beginning of file through first “<dict>” declaration to a new TextWrangler file
3. Search for “ShowHardDrivesOnDesktop” – copy both the “<key>…</key>” section and “<true/>” or “<false/>” values to the new file
4. Close out the file with “</dict></plist>” and save to your desktop as “com.apple.finder.plist”
5. Convert the new .plist file to a mobileconfig with McxToProfile
   1. /path/to/mcxToProfile --plist=/path/to/new/com.apple.finder.plist \  
      --identifier=edu.syr.training.desktopsettings \  
      --displayname=”Stupid Tricks Demo”
6. Double click resultant .mobileconfig file to apply it to the system
7. Mobileconfig configuration profiles can also be applied with the ‘profiles’ command:  
    $ profiles –I –F /path/to/studidTricksDemo.mobileconfig

# Packaging

Apple’s OS X Installation packages can contain files, applications and scripts that run during the installation process. These installations leave records behind in the system receipts database, which can be inspected

Two utilities addressed today: pkgbuild and pkgutil

1. pkgbuild: used to create and optionally sign installation package.
2. pkgutil: used to expand or re-wrap flat package archives, inspect contents, and interact with the receipts database.

Installation packages can be easily built using a “destination root” approach. With this technique, applications and files can be placed within folders that are nested under a “root” folder to mimic their destination path on the target drive.

Installation scripts can be collected into a directory which will be specified when building the package. Apple has two reserved names (preinstall & postinstall) that will fire exactly when you would expect given their names. So if you wish to execute a script that does something with a file you have just copied (apply a configuration profile, for instance), it MUST be named “postinstall” and be placed into the directory that you will specify as the scripts directory.

To build an installation package with the provided examples, you would use the following command (assuming you have ‘cd’d to the packaging folder - otherwise full paths will be needed for the root and scripts arguments):

$ pkgbuild --root ./root --scripts ./scripts --identifier edu.syr.training.desktoppicture --version 1.0 --filter “\.DS\_Store” DesktopPictureExample-1.0.pkg

So, let’s break that down:

* pkgbuild calls the command
* --root specifies the base of the destination root folder that we have created
* --scripts specifies where to locate any installer scripts
* --identifier specifies the name of the package receipt that will be recorded in the receipts database
* --version because every installation has to have a version
* --filter excludes specific files. The filter command accepts POSIX regular expression formats (see “man re\_format” for details), so we have to escape the “.” char in order to look for it specifically. This command may be used multiple times.
* And finally we specify the filename for the resulting install package.

It is worth noting that you can create an installation package that does nothing but execute a script. To do this, you can completely skip the “root” and “filter” portions of the above command, but you will need to add the “--nopayload” flag.