Automating OS X and iOS Configuration

acmefoo!

Scott M. Neal
smn.MG@acmefoo.org

MacTech 2011

Automating Configuration: Agenda

Property Lists

- brief review

Defaults

- Domains
- Accessing Defaults from the CLI
  - `defaults`
  - `PlistBuddy`

MCX

- Organization
- Accessing MCX from
  - GUI
  - CLI

Configuration Profiles

- iOS vs. Mac OS X
- iPhoneConfigurationUtility
- Profile Manager (OS X Lion Server GUI)

Short preso--hopefully enough to plant seeds for automations

Property Lists

Property Lists

Configuration information for processes needs to be stored somewhere

Apple (actually NeXT) developed the **Property List** (aka **plist**) Format to provide a consistent way to store information

- Not ALL configuration information is stored in plist format

Property Lists provide the backbone for the **Defaults** Preference infrastructure

Property Lists: Key-Value

Property Lists are **Key-Value** storage mechanisms

- The **Key** is the identifier for a particular property
- The **Value** is the storage container referenced by its Key

Example:

```
name = “AppleScript”
```

- the Key is `name`
- the Value of the Key `name` is `AppleScript`

Property Lists: Value Classes

Values can belong to different **Value Classes**

- **String**
  - A collection of characters
  - Keys themselves are strings
- **Number**
  - A numeric value that can participate in calculations
- **Boolean**
  - True/False, Yes/No, 1/0
- **Date**
  - A specific numeric type used to store dates
- **Data**
  - Raw data stored in hexadecimal

Property Lists: Value Classes (cont.)

There are two types of Value Classes that are used for organizing other Value Classes

- **Array**
  - An ordered list of objects, numbered from item 0 (the first item) to n (the last item)
- **Dictionary**
  - A Key-Value group, where the Key is a string (as normal), and the Value can be ANY Value Class type

These are known as **Container Classes**

Property List Formats

There are currently 3 Property List file formats:

- **XML**
- **Text** (historic NeXT-style)
  - formerly called ASCII (incorrectly!)
- **Binary**

Property Lists are stored with the file extension **.plist**

The Property List infrastructure can read/write all 3 formats

- Humans can read/write the first 2 formats

Property List Formats: XML

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/
PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
         <key>DateFormat</key>
         <string>E h:mm a</string>
         <key>FlashDateSeparators</key>
         <false/>
         <key>IsAnalog</key>
         <false/>
</dict>
</plist>
```

Property List Formats: Text/NeXT

```
{
    DateFormat = "E h:mm a";
    FlashDateSeparators = 0;
    IsAnalog = 0;
}
```

Property List Formats: Binary

```
bplist00?ZDateFormatXIsAnalog_FlashDateSeparatorsXE h:m#9BCD
```

Editing Property Lists

Text Editor (TextEdit, vi, nano, etc.)

- Doesn’t work with Binary (unless plist is converted to one of the other formats)

Property List Editor

- Located in /Developer/Applications/Utilities
- Works with ALL plist formats
- Included with some versions of OS X Server--no need to install developer tools

Converting from one plist format to another

“Save To” option of Property List Editor

command-line `plutil` tool

Property Lists: Summary

YOU are now empowered to edit Property List files directly

- Double-edged sword: you can also SCREW UP a process' configuration, making your process (or even entire machine) unusable, so PLEASE backup plist files and edit copies!
- Programmers don't tend to publish "how to" manuals
  - And they tend to completely change things
- Great for troubleshooting/debugging
  - Avoid "Trashing the preferences" (which really should be "Renaming the preferences")
  - single user mode
- Can be utilized for your own scripts

Property List Locations

OS X has a standard hierarchical resource search policy (for Fonts, etc.):

    ~/Library/*
    /Library/*
    /Network/Library/* (if it exists)
    /System/Library/*

Property Lists are typically, but not necessarily searched hierarchically and can be stored ANYWHERE, typically:

- /etc/*
- Application Bundles
- Preferences (but where?)
    - remember, not all plist files are for preferences

Defaults

Preference Files: Defaults

The **Defaults** system is a Preference storage organization infrastructure

- Analogous to Windows Registry

Uses Property Lists

- Keys in each plist are specific to each process and define configuration options for that process, such as
  - Default new window location and size
  - File location/path
  - whatever the programmer decides is important to save as defaults: the sky is the limit...

It is possible to modify preferences by:

- Editing a Preference plist file directly
  - like we have seen already...
- Using the Defaults system
  - Better for Automation

Preferences: Defaults Infrastructure

Defaults is a “portal” into the Preferences plist system

- It does NOT incorporate EVERY plist on the system (phew!), only the ones dedicated to preferences and stored in specific folders
- Processes themselves don’t need to understand how to read/write or find plist files, they use the Defaults infrastructure

Your own scripts can utilize the Defaults system to store their own information

Defaults Infrastructure

Location of Preference files:

- ~/Library/Preferences
- /Library/Preferences
- /var/root/Library/Preferences

Most files in these folders are in plist format

Note that some files in */Preferences are NOT plist format

- These will not be accessible through the Defaults system, but are read directly by the process that owns that preference

Defaults Domains

Preferences in the Defaults system are organized by domain (and, optionally, host)

- Typically correspond to individual applications/ processes

Nomenclature for these domains typically (not always) follows Java reverse-FQDN syntax WITHOUT the .plist extension

- Prevents namespace collisions
- Examples:

  com.apple.dock
  com.adobe.versioncue
  com.apple.loginwindow
  loginwindow

Accessing Defaults

Defaults are accessed using the CLI tool `defaults`

- When a "relative" Domain is specified, `defaults` searches ~/Library/Preferences for a plist file matching the Domain argument with .plist appended

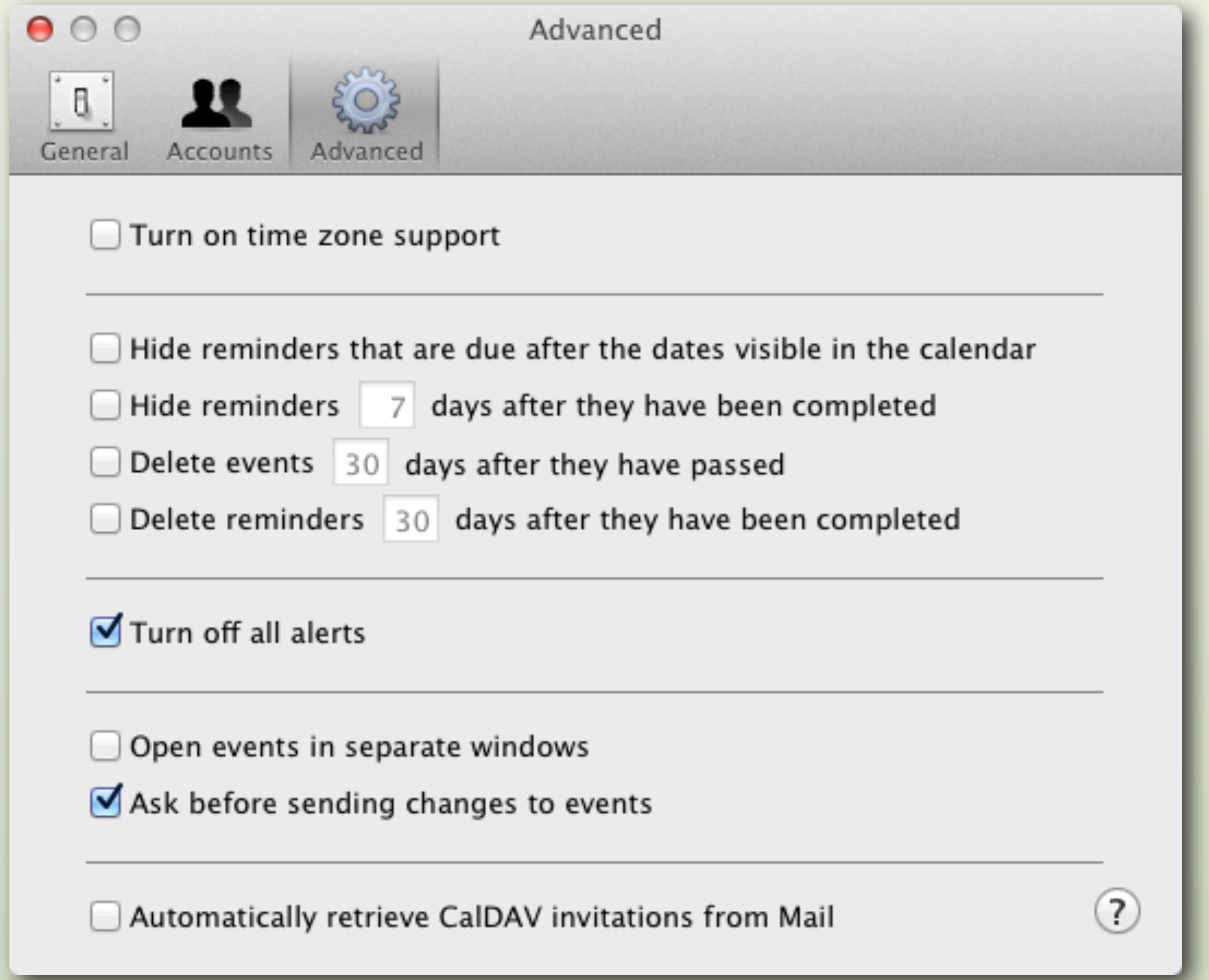
Example:

```
defaults read com.apple.iCal
```

Note that output from `defaults` is in Text/NeXT format, independent of the plist format itself (Binary, XML, or Text/NeXT)

Domain com.apple.iCal

```
{
    CalDefaultCalendar = "CEA6202C-B0BE-436D-A214-26B774634825";
    CalDefaultPrincipal = CalCalendarLocalGroupIdentifier;
    CalDefaultReminderList = "14D59E7F-F889-4E03-95A4-ED922F70E0E9";
    CalSuccessfulLaunchTimestampPreferenceKey = "3.416863e+08";
    DelegatesInSeparateWindows =     {
        iCal =         {
        };
    };
    DeleteExpiredTodos = 0;
    "Disable all alarms" = 1;
    DisableEmphasizedViews = 1;
    NSDontMakeMainWindowKey = NO;
...
    "TimeZone support enabled" = 0;
    "delete todos after" = 30;
    "first minute of day time range" = 0;
    "first shown minute of day" = 452;
    "iCal version" = 83886080;
    "last minute of day time range" = 1440;
    lastViewsTimeZone = "America/Los_Angeles";
    "view rects" =     {
        "1-day" = "{{50, 50}, {600, 400}}";
        "2-day" = "{{50, 50}, {600, 400}}";
        "3-day" = "{{50, 50}, {600, 400}}";
        "4-day" = "{{50, 50}, {600, 400}}";
        "5-day" = "{{50, 50}, {700, 500}}";
        "6-day" = "{{50, 50}, {700, 500}}";
        "7-day" = "{{50, 50}, {770, 500}}";
        monthly = "{{50, 50}, {750, 550}}";
        yearly = "{{50, 50}, {750, 550}}";
    };
}
```



- [ ] Turn on time zone support
- [ ] Hide reminders that are due after the dates visible in the calendar
- [ ] Hide reminders 7 days after they have been completed
- [ ] Delete events 30 days after they have passed
- [ ] Delete reminders 30 days after they have been completed
- [✓] Turn off all alerts
- [ ] Open events in separate windows
- [✓] Ask before sending changes to events
- [ ] Automatically retrieve CalDAV invitations from Mail

Reading/Writing Values for Specific Keys

A specific Key can be read using `defaults`

```
defaults read com.apple.iCal \
DeleteExpiredTodos
```

The following examples “do the same thing”

- A Key’s Value can be modified:

```
defaults write com.apple.iCal \
DeleteExpiredTodos 0
```

- A specific Value Class can be specified:

```
defaults write com.apple.iCal \
DeleteExpiredTodos -boolean false
```

Another Example

```
defaults read com.apple.iCal \
“delete todos after”
```

Other Defaults Domains

`NSGlobalDomain`

- Used for default shared Key-Value combos which are not process, host, or domain-specific
- User-specific
  - We will see how to get global shared defaults
- Can use `-g` instead of specifying `NSGlobalDomain`
- Examples:

```
defaults read NSGlobalDomain \
AppleMiniaturizeOnDoubleClick

defaults read -g AppleMiniaturizeOnDoubleClick
```

Other Defaults Domains (cont.)

A full path to a .plist file (minus the .plist extension) can be specified as a domain

- Example:

```
defaults read /Library/Preferences/\
com.apple.loginwindow
```

Take a look at the Domains available in each of the Preference file locations:

- ~/Library/Preferences
- /Library/Preferences
- /var/root/Library/Preferences

Specifying a Host with Defaults

The `ByHost` folder seen in some of the previous Preference locations stores host-specific information

- Useful for processes that utilize more than one host, and want to have host-specific preferences
- Unique identification via
  - machine's MAC (hardware) address
  - UUID
- Examples:
  - Screen Saver
  - Time Machine
  - Printers

Specifying a Host with Defaults

You may specify a hostname as

- `-currentHost`

```
defaults -currentHost read \
com.apple.loginwindow
```

- MAC (IP) address

```
defaults -host 000a95a92943 read
```

- UUID (Leopard and beyond)

```
defaults -host 1178616B-6E68-5BAA-BDA0-0A05905571DD \
read
```

Global Preferences

User-level, not host-specific

`~/Library/Preferences/.GlobalPreferences.plist`

- This is an FYI, since you really should be using **`-g`** or **`NSGlobalDomain`** to edit a user's global data

User-level, host-specific

`~/Library/Preferences/ByHost/.GlobalPreferences.`***`MacAddress`***`.plist`

or

`~/Library/Preferences/ByHost/.GlobalPreferences.`***`UUID`***`.plist`

System

`/Library/Preferences/.GlobalPreferences.plist`

Domain Synopsis

Search Order	User Scope	App Scope	Host Scope	File System Location
1	Specific	Specific	Specific	~/Library/Preferences/ByHost/*appfqdn*.*macaddress*.plist
2	Specific	Specific	Any	~/Library/Preferences/*appfqdn*.plist
3	Specific	Any	Specific	~/Library/Preferences/ByHost/.GlobalPreferences.*macaddress*.plist
4	Specific	Any	Any	~/Library/Preferences/.GlobalPreferences.plist
5	Any	Specific	Specific	/Library/Preferences/ByHost/*appfqdn*.*macaddress*.plist
6	Any	Specific	Any	/Library/Preferences/*appfqdn*.plist
7	Any	Any	Specific	/Library/Preferences/ByHost/.GlobalPreferences.*macaddress*.plist
8	Any	Any	Any	/Library/Preferences/.GlobalPreferences.plist

Accessing `defaults` by Application Name

It is often easier to access an application's preferences by application name:

```
defaults read -app TextEdit
```

Accessing defaults via AppleScript

Can use defaults infrastructure directly from within AppleScript

- same limitations as `defaults` command from CLI

Can write Automations to access current defaults, or have your own Automations have stored preferences

- how cool is THAT?

Utilizes AppleScript/ObjC

- Different syntax (similar) for older AppleScript Studio

Accessing defaults via AppleScript

Creating new entry

```
-- set up initial default properties
property importantValue : "MacTech2011"

on applicationWillFinishLaunching_(aNotification)
-- set up initial defaults
tell standardUserDefaults() of NSUserDefaults ¬
registerDefaults_({importantValue: importantValue})
```

Accessing defaults via AppleScript

Setting value of pre-existing default

```
tell standardUserDefaults() of NSUserDefaults
    setObject_forKey_("MacTech2012", "importantValue")
end tell
```

Reading value

```
tell standardUserDefaults() of NSUserDefaults
    set whatToDo to objectForKey_("importantValue")
end tell
```

By default, stored in application domain

- .plist file is named using the application identifier
    - Not seen in these slides, but would be in Xcode project
- Can use other domains

Accessing defaults via Other scripting

Utilize whatever the native scripting environment's method is to execute standard commands

```
#!/usr/bin/env perl

$myAppDomain = "com.apple.iCal";
$domainPath = "~/Library/Preferences/" .
$myAppDomain;
$command = "defaults read " . $domainPath .
" \"iCal version\"";
$result = `$command`;
print ($result . "\n");
```

Accessing defaults via Other scripting

Utilize whatever the native scripting environment's method is to execute standard commands

```
#!/usr/bin/env perl

$myAppDomain = "org.acmefoo.coolproject";
$domainPath = "~/Library/Preferences/" .
$myAppDomain;
$command = "defaults read " . $domainPath .
" importantValue";
$result = `$command`;
print ($result . "\n");
```

PlistBuddy

`defaults` Limitations

The defaults command can only edit top-level of a plist

`defaults` Limitations

The defaults command can only edit top-level of a plist

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/
PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
...
         <key>CalSuccessfulLaunchTimestampPreferenceKey</key>
         <real>341686272</real>
         <key>DelegatesInSeparateWindows</key>
         <dict>
               <key>iCal</key>
               <dict/>
          </dict>
          <key>DeleteExpiredTodos</key>
          <true/>
         <key>PersistentMenu-lastUsedTimeZones</key>
         <array>
               <dict>
               <key>name</key>
               <string>America/Los_Angeles</string>
               </dict>
         </array>
...
</dict>
</plist>
```

defaults Limitations

The defaults command can only edit top-level of a plist

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/
PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
...
         <key>CalSuccessfulLaunchTimestampPreferenceKey</key>
         <real>341686272</real>
         <key>DelegatesInSeparateWindows</key>
         <dict>
               <key>iCal</key>
               <dict/>
          </dict>
          <key>DeleteExpiredTodos</key>
          <true/>
         <key>PersistentMenu-lastUsedTimeZones</key>
         <array>
               <dict>
               <key>name</key>
               <string>America/Los_Angeles</string>
               </dict>
         </array>
...
</dict>
</plist>
```

`defaults` Limitations

The defaults command can only edit top-level of a plist

```
defaults write DeleteExpiredTodos \
DeleteExpiredTodos -boolean false
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/
PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
...
         <key>CalSuccessfulLaunchTimestampPreferenceKey</key>
         <real>341686272</real>
         <key>DelegatesInSeparateWindows</key>
         <dict>
               <key>iCal</key>
               <dict/>
          </dict>
          <key>DeleteExpiredTodos</key>
          <true/>
         <key>PersistentMenu-lastUsedTimeZones</key>
         <array>
               <dict>
               <key>name</key>
               <string>America/Los_Angeles</string>
               </dict>
         </array>
...
</dict>
</plist>
```

`defaults` Limitations

The defaults command can only edit top-level of a plist

```
defaults write DeleteExpiredTodos \
DeleteExpiredTodos -boolean false
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/
PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
...
         <key>CalSuccessfulLaunchTimestampPreferenceKey</key>
         <real>341686272</real>
         <key>DelegatesInSeparateWindows</key>
         <dict>
               <key>iCal</key>
               <dict/>
          </dict>
          <key>DeleteExpiredTodos</key>
          <false/>
         <key>PersistentMenu-lastUsedTimeZones</key>
         <array>
               <dict>
               <key>name</key>
               <string>America/Los_Angeles</string>
               </dict>
         </array>
...
</dict>
</plist>
```

`defaults` Limitations

The defaults command can only edit top-level of a plist

```
defaults write DeleteExpiredTodos \
DeleteExpiredTodos -boolean false
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/
PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
...
         <key>CalSuccessfulLaunchTimestampPreferenceKey</key>
         <real>341686272</real>
         <key>DelegatesInSeparateWindows</key>
         <dict>
               <key>iCal</key>
               <dict/>
          </dict>
          <key>DeleteExpiredTodos</key>
          <false/>
         <key>PersistentMenu-lastUsedTimeZones</key>
         <array>
               <dict>
               <key>name</key>
               <string>America/Los_Angeles</string>
               </dict>
         </array>
...
</dict>
</plist>
```

So Far, So Good

`defaults` Limitations

The defaults command can only edit top-level of a plist

defaults Limitations

The defaults command can only edit top-level of a plist

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/
PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
...
         <key>CalSuccessfulLaunchTimestampPreferenceKey</key>
         <real>341686272</real>
         <key>DelegatesInSeparateWindows</key>
         <dict>
               <key>iCal</key>
               <dict/>
          </dict>
          <key>DeleteExpiredTodos</key>
          <false/>
         <key>PersistentMenu-lastUsedTimeZones</key>
         <array>
               <dict>
               <key>name</key>
               <string>America/Los_Angeles</string>
               </dict>
         </array>
...
</dict>
</plist>
```

`defaults` Limitations

The defaults command can only edit top-level of a plist

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/
PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
...
         <key>CalSuccessfulLaunchTimestampPreferenceKey</key>
         <real>341686272</real>
         <key>DelegatesInSeparateWindows</key>
         <dict>
               <key>iCal</key>
               <dict/>
          </dict>
          <key>DeleteExpiredTodos</key>
          <false/>
         <key>PersistentMenu-lastUsedTimeZones</key>
         <array>
               <dict>
               <key>name</key>
               <string>America/Los_Angeles</string>
               </dict>
         </array>
...
</dict>
</plist>
```

`defaults` Limitations

The defaults command can only edit top-level of a plist

```
defaults write com.apple.iCal \
PersistentMenu-lastUsedTimeZones      ...?
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/
PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
...
         <key>CalSuccessfulLaunchTimestampPreferenceKey</key>
         <real>341686272</real>
         <key>DelegatesInSeparateWindows</key>
         <dict>
               <key>iCal</key>
               <dict/>
          </dict>
          <key>DeleteExpiredTodos</key>
          <false/>
         <key>PersistentMenu-lastUsedTimeZones</key>
         <array>
               <dict>
               <key>name</key>
               <string>America/Los_Angeles</string>
               </dict>
         </array>
...
</dict>
</plist>
```

`defaults` Limitations

The defaults command can only edit top-level of a plist

```
defaults write com.apple.iCal \
PersistentMenu-lastUsedTimeZones      ...?
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/
PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
...
         <key>CalSuccessfulLaunchTimestampPreferenceKey</key>
         <real>341686272</real>
         <key>DelegatesInSeparateWindows</key>
         <dict>
               <key>iCal</key>
               <dict/>
          </dict>
          <key>DeleteExpiredTodos</key>
          <false/>
         <key>PersistentMenu-lastUsedTimeZones</key>
         <array>
               <dict>
               <key>name</key>
               <string>America/Los_Angeles</string>
               </dict>
         </array>
...
</dict>
</plist>
```

We can either:

- Replace array completely
- Add item to end of array

We cannot:

- Delete an item from array
- Modify an item in-place

Same rules apply to Dictionaries

To the Rescue: `PlistBuddy`

Formerly a hard-to-find (and barely acknowledged by Apple) CLI tool

- Currently located at `/usr/libexec/PlistBuddy`
- Has a very well-written `man` page

Has both interactive and non-interactive modes

MUCH more powerful (but a bit more complicated) than `defaults`

Value Class types available from `PlistBuddy`:

- `string`
- `real`
- `integer`
- `bool`
- `date`
- `data`
- `array`
- `dict`

`PlistBuddy` Basics

`PlistBuddy` has no cognizance of Domains, you just give it the full path (including the .plist) to the Plist file you'd like to edit

`PlistBuddy` traverses hierarchy within a plist using an entry

- From the man page:
  - Entries consist of property key names delimited by colons. Array items are specified by a zero-based integer index. Examples:
    - :CFBundleShortVersionString
    - :CFBundleDocumentTypes:2:CFBundleTypeExtensions
- Notice the use of ':' to delimit the key path
  - Ahhh, the nostalgia of using a colon as a path delimiter...

*** WARNING ***

- If you would like to follow along, remember that you will be modifying live files on your computer
    - Please do so at your own risk!

Using `PlistBuddy`

Showing the contents of a plist (no matter what format it is stored in):

```
/usr/libexec/PlistBuddy -c \
"Print" \
~/Library/Preferences/com.apple.iCal.plist
```

(equivalent to the following with defaults)

```
defaults read com.apple.iCal
```

Mimicking

```
defaults write com.apple.iCal \
DeleteExpiredTodos -boolean false
```

using `PlistBuddy`:

```
/usr/libexec/PlistBuddy -c \
"Set :DeleteExpiredTodos false" \
~/Library/Preferences/com.apple.iCal.plist
```

Using `PlistBuddy`

Removing an item from an array:

```
/usr/libexec/PlistBuddy -c \
"Delete :PersistentMenu-lastUsedTimeZones:3" \
~/Library/Preferences/com.apple.iCal.plist
```

Replacing an item in an array:

```
/usr/libexec/PlistBuddy -c \
"Set : PersistentMenu-lastUsedTimeZones:0:name \
America/Los_Angeles" \
~/Library/Preferences/com.apple.iCal.plist
```

Printing a nested item:

```
/usr/libexec/PlistBuddy -c \
"Print :PersistentMenu-lastUsedTimeZones:0:name"
~/Library/Preferences/com.apple.iCal.plist
```

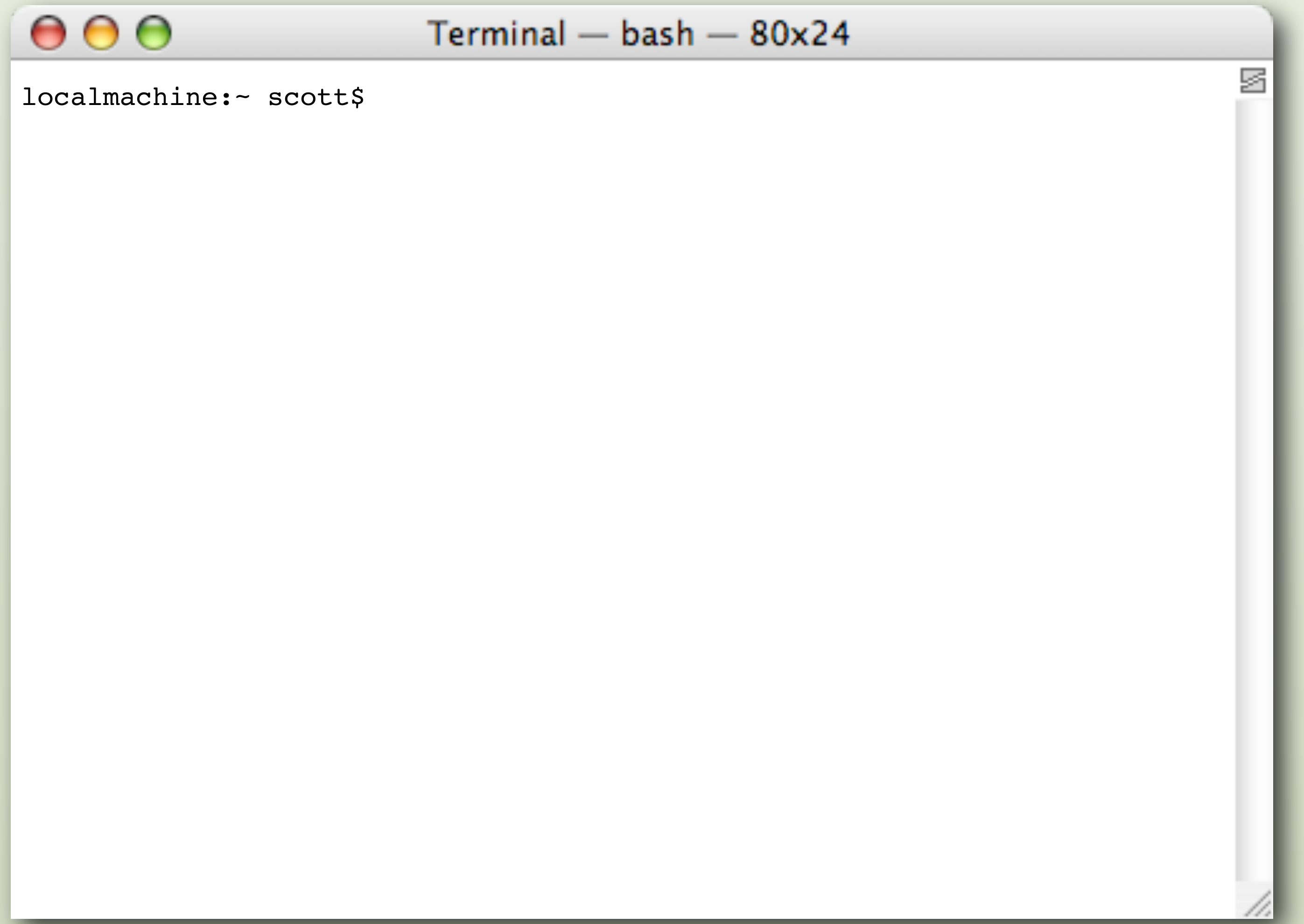
Using `PlistBuddy` in Interactive Mode

Great place to practice syntax before putting into Automation!

When making changes, don't forget to Save...

Using `PlistBuddy` in Interactive Mode

- Great place to practice syntax before putting into Automation!
- When making changes, don't forget to Save...

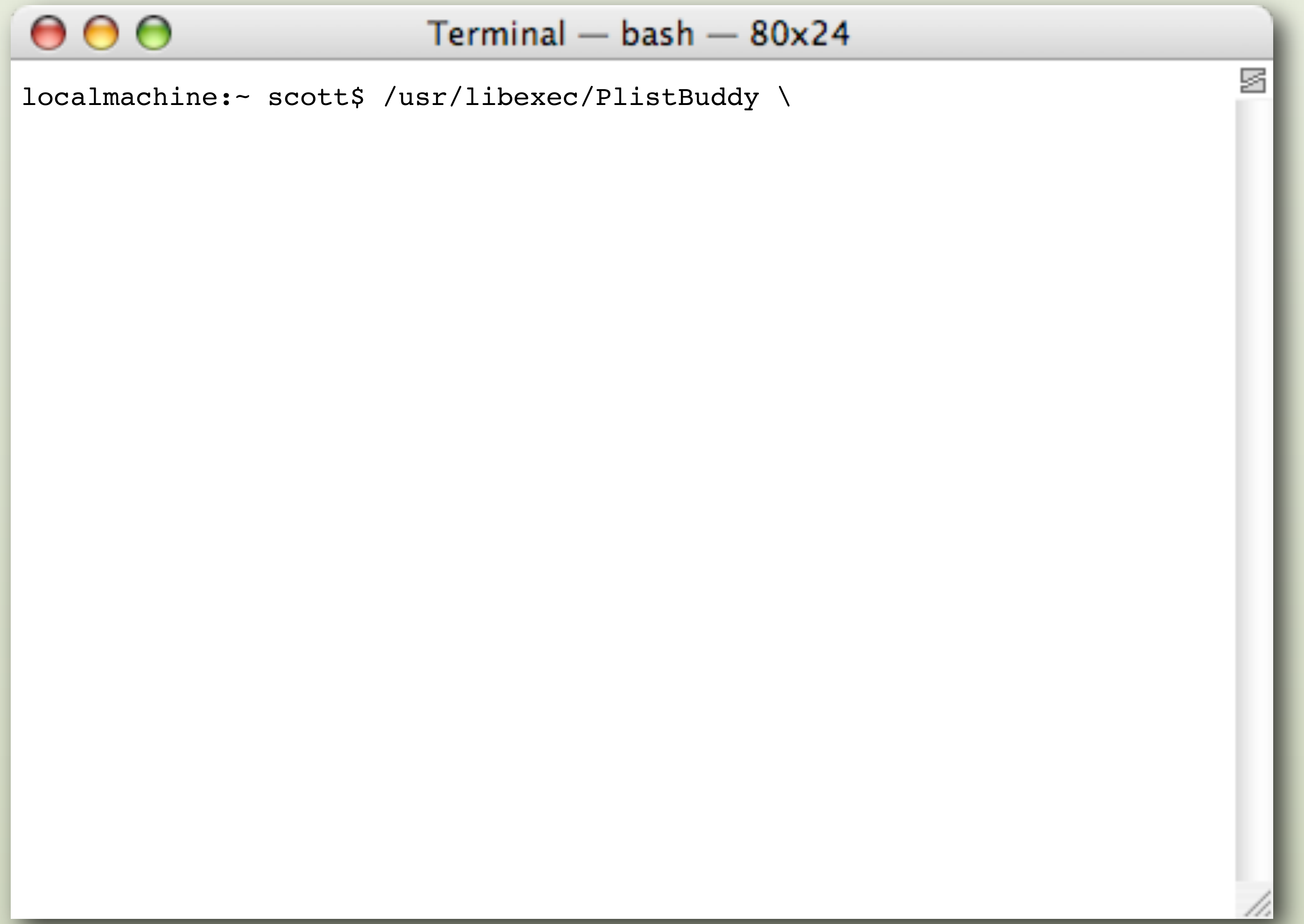


```
localmachine:~ scott$
```

Using `PlistBuddy` in Interactive Mode

Great place to practice syntax before putting into Automation!

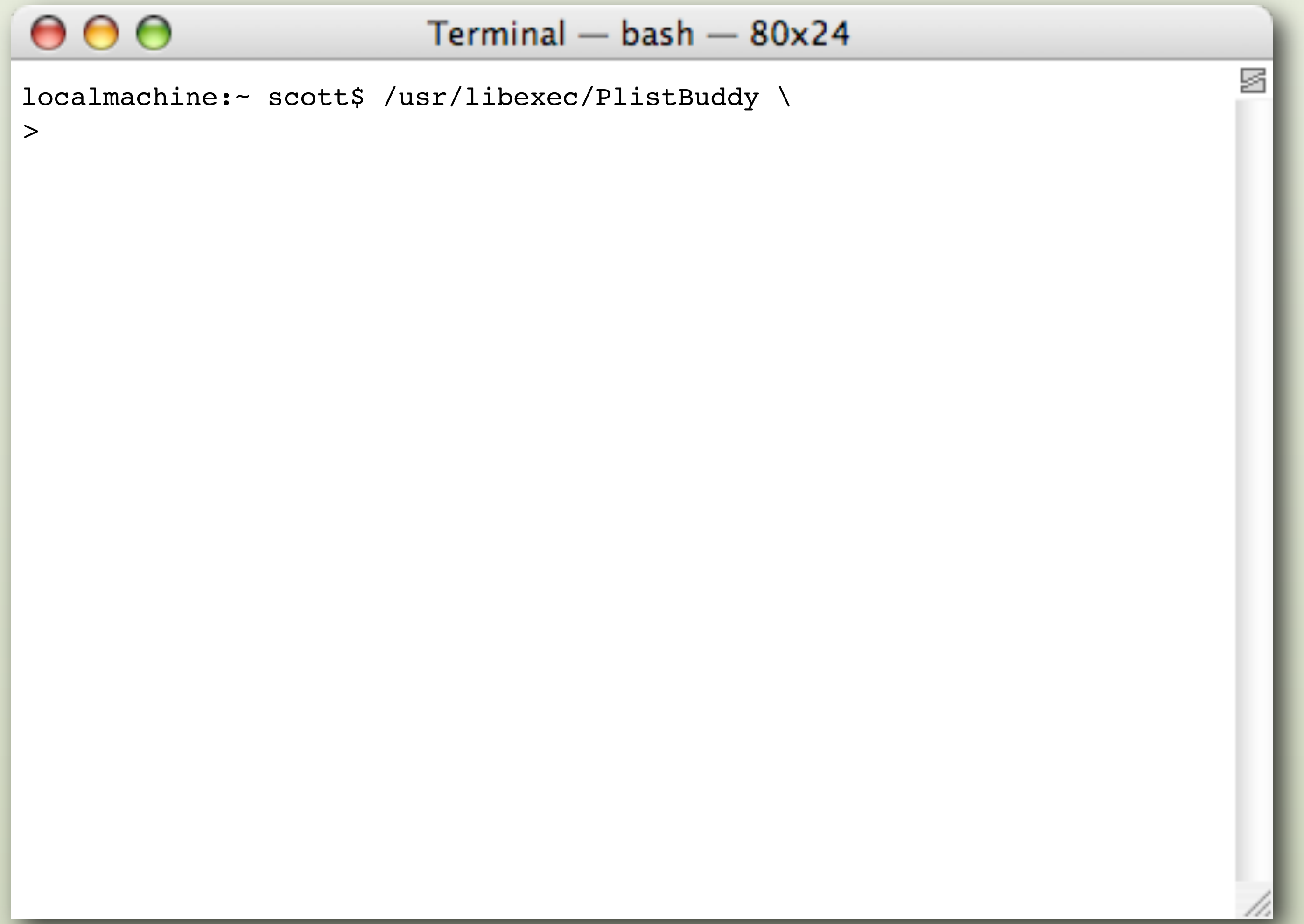
When making changes, don't forget to Save...



```
localmachine:~ scott$ /usr/libexec/PlistBuddy \
```

Using `PlistBuddy` in Interactive Mode

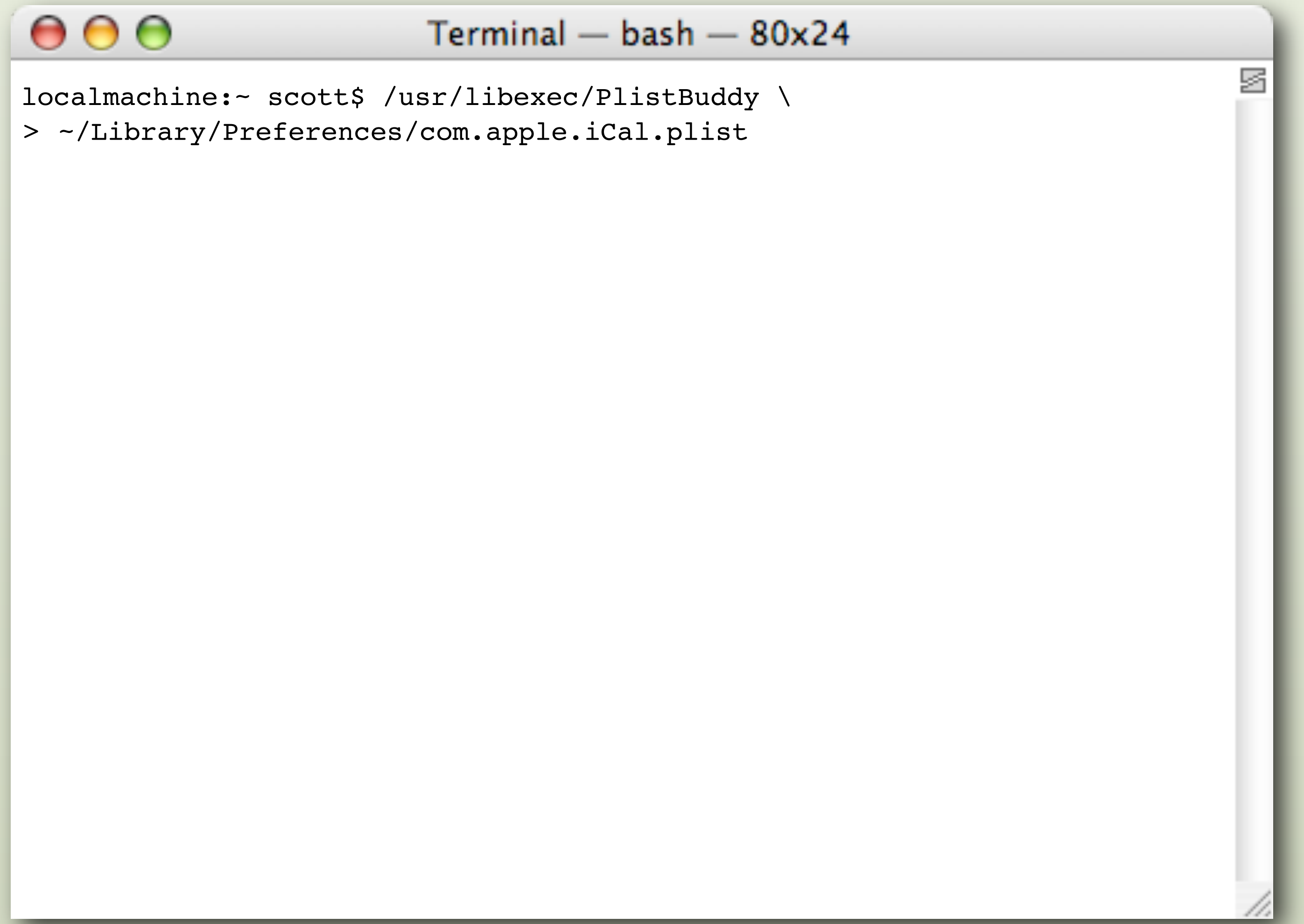
- Great place to practice syntax before putting into Automation!
- When making changes, don't forget to Save...


Terminal — bash — 80x24

```
localmachine:~ scott$ /usr/libexec/PlistBuddy \
>
```

Using `PlistBuddy` in Interactive Mode

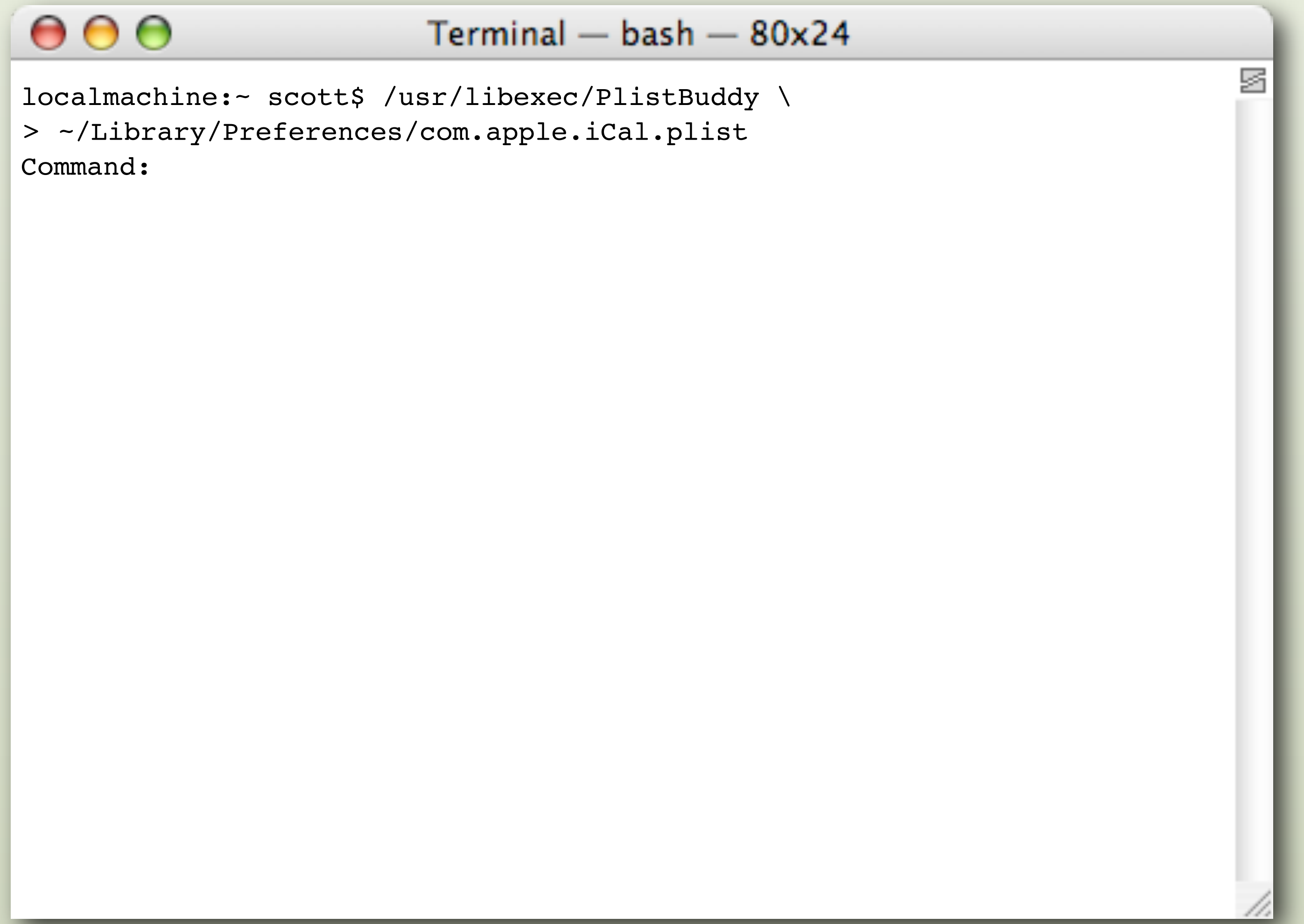
- Great place to practice syntax before putting into Automation!
- When making changes, don't forget to Save…



```
localmachine:~ scott$ /usr/libexec/PlistBuddy \
> ~/Library/Preferences/com.apple.iCal.plist
```

Using `PlistBuddy` in Interactive Mode

- Great place to practice syntax before putting into Automation!
- When making changes, don't forget to Save...



```
localmachine:~ scott$ /usr/libexec/PlistBuddy \
> ~/Library/Preferences/com.apple.iCal.plist
Command:
```

PlistBuddy

If the info you wish to view/use/change in your Automation is in a plist, you have complete control

- CLI
- Any scripting environment that can access CLI

For some hopefully useful examples:

- http://www.afp548.com/forum/viewtopic.php?showtopic=23851
- http://explanatorygap.net/2006/07/14/plistbuddy-is-the-shiznit/
- http://macscripter.net/viewtopic.php?id=18380

MCX: Managed Client on Mac OS X

MCX

Main method of client management previous to Mac OS X Server 10.7 Lion

Configured via

- GUI tools
- CLI tools

Information usually distributed to clients via OD

- Then cached in local directory services and defaults

MCX isn’t it’s own category, but a set of attributes added to the standard OD categories

- User
- WorkGroup
- Computer

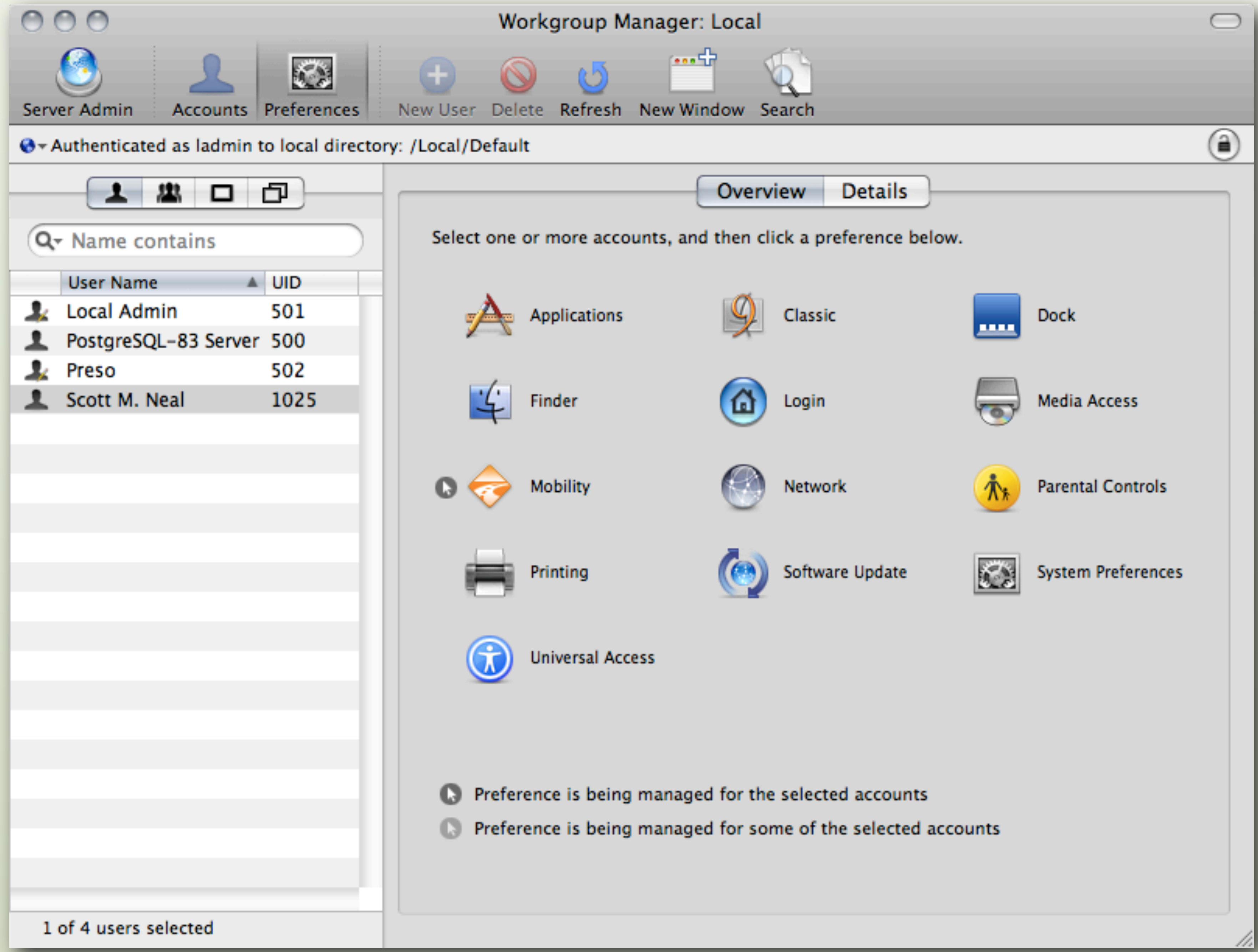
Many Apple applications utilize MCX

- Custom applications can utilize Preference Manifests

MCX: GUI

Mac OS X Server

- WorkGroup Manager



Authenticated as ladmin to local directory: /Local/Default

User Name	UID
Local Admin	501
PostgreSQL-83 Server	500
Preso	502
Scott M. Neal	1025

1 of 4 users selected

Select one or more accounts, and then click a preference below.

Applications, Classic, Dock, Finder, Login, Media Access, Mobility, Network, Parental Controls, Printing, Software Update, System Preferences, Universal Access

Preference is being managed for the selected accounts

Preference is being managed for some of the selected accounts

MCX: Dedicated CLI

Easier than digging using `defaults/PlistBuddy`

`dscl`

- Syntax
  - `-mcxread` *`recordPath`* `[-v` *`mcxVersion`*`] [-o` *`filePath`*`] [-format {xml | plist | text}] [`*`appDomain`* `[`*`keyName`*`]]`
  - `-mcxset` *`recordPath`* `[-v` *`mcxVersion`*`]` *`appDomain`* *`keyName`* `[`*`mcxDomain`* `[`*`keyValue`* `[`*`UPK`*`]]]`
  - `-mcxedit` *`recordPath`* `[-v` *`mcxVersion`*`]` *`appDomain`* *`keyPath`* `[`*`keyValue`*`]`
  - `-mcxdelete` *`recordPath`* `[-v` *`mcxVersion`*`] [`*`appDomain`* `[`*`keyName`*`]]`
  - `-mcxexport` *`recordPath`* `[-o` *`filePath`*`] [-format {xml | plist | text}] [`*`appDomain`* `[`*`keyName`*`]]`
  - `-mcximport` *`recordPath`* `[-d]` *`filePath`*
  - `-mcxhelp`

MCX: CLI

`mcxquery`

- Display managed preferences for a user/workgroup/computer/computergroup combination
- Example
  - `mcxquery -user jane -group science -computer lab1_12`

`mcxrefresh`

- Preferences are only “loaded” when a user logs in
- Use this command to manually reload
- Example (typos in Mac OS X Server 10.6 User Mgmt doc...)
  - `mcxrefresh -n 'ajohnson'`

Configuration Profiles

Configuration Profiles

Configuration profiles are XML files used for configuring Mac OS X and iOS devices

- End with suffix .mobileconfig (iOS lineage)

Rather than users/workgroups/computers/computergroups:

- User (groups)
- Device (groups)

Distribution methods

- Manual
- User Portal
- Remote Device Management

Payloads are installed all-or-nothing

- If a single profile contains configuration for VPN and restrictions, can't install one without the other

Tools for Creating Configuration Profiles

iPhone/iOS Configuration Utility

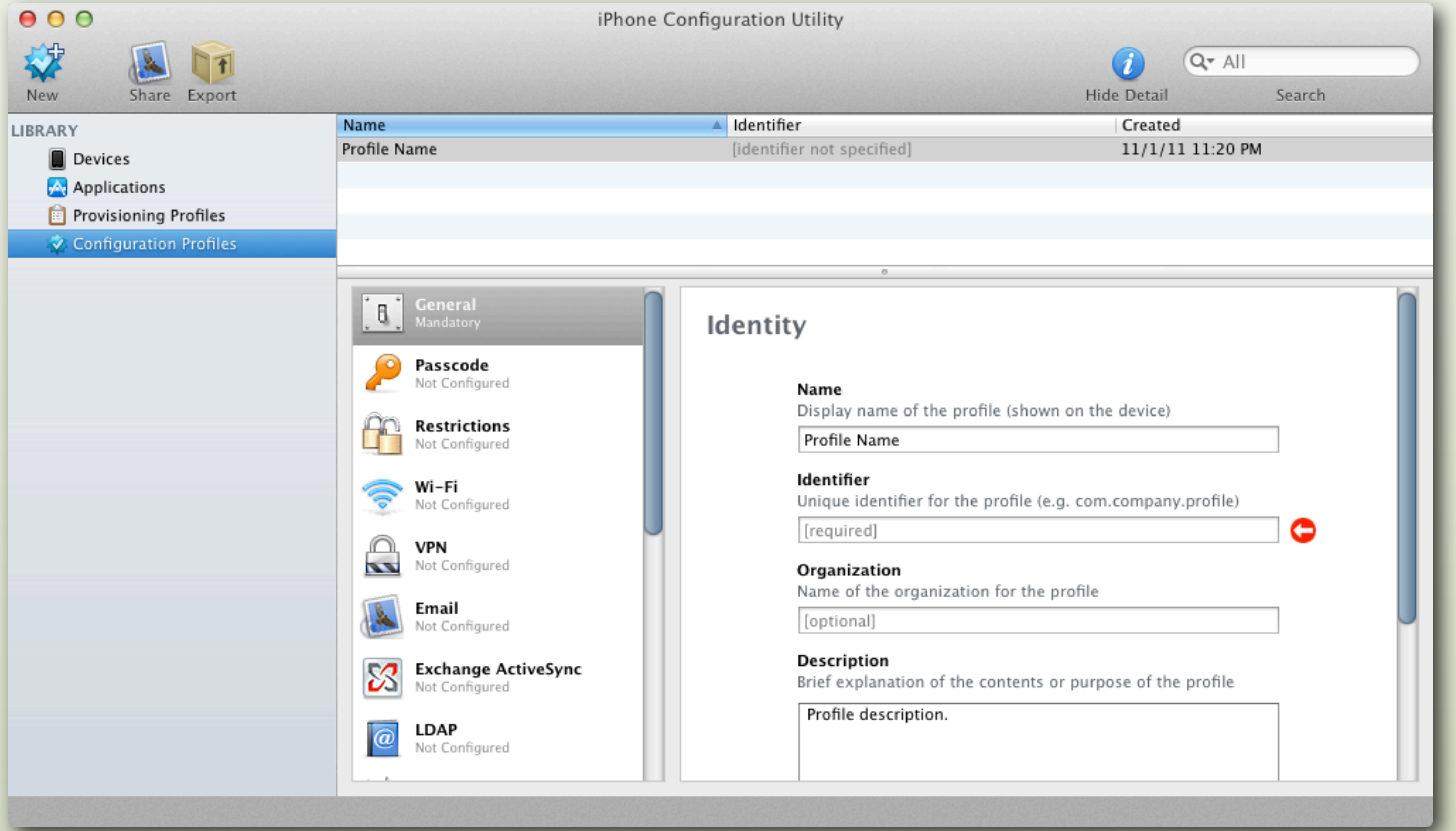
- sole option for pre-Lion (Leopard, Snow Leopard)
- Windows XP/Vista
- http://support.apple.com/kb/dl851

Profile Manager

- Mac OS X Server 10.7 (Lion)
- http://help.apple.com/profilemanager/mac/10.7

Mac OS X Snow Leopard iPhoneConfigurationUtility

AppleScriptable



Name	Identifier	Created
Profile Name	[identifier not specified]	11/1/11 11:20 PM

General
Mandatory

Passcode
Not Configured

Restrictions
Not Configured

Wi-Fi
Not Configured

VPN
Not Configured

Email
Not Configured

Exchange ActiveSync
Not Configured

LDAP
Not Configured

Identity

Name
Display name of the profile (shown on the device)
Profile Name

Identifier
Unique identifier for the profile (e.g. com.company.profile)
[required]

Organization
Name of the organization for the profile
[optional]

Description
Brief explanation of the contents or purpose of the profile
Profile description.

Mac OS X Server Lion Profile Manager

Supports push-delivered profile delivery and RDM

Devices

- Name
- ID
  - Serial Number
  - Unique Device IDentifier (UDID)
  - Mobile Equipment Identity (IMEI)
    - iPad 3G, ...?
  - Mobile Equipment Identifier (MEID)
    - Verizon

Mac OS X Lion CLI

`profiles`

- Available in Mac OS X Lion
  - Snow Leopard has a command “profiles” but it is for SAMBA

`mdmclient`

- not meant to be run from CLI--backend process

Configuration Profile Delivery: Manual

Configuration profiles can be emailed, or put onto a sharepoint, or whatever you want

Users find these profiles, download them, and install

Configuration Profile Delivery: User Portal

Via OS X Server Lion web portal, users can log in and get configuration profiles

- Profile Manager Install Portal

Configuration Profile Delivery: Remote

Remote Device Management (RDM) & Mobile Device Management (MDM)

- via Apple's Push servers
- Must be pre-config'ed

Summary

Defaults

- Preferences are managed through the Defaults infrastructure, and stored in plist files in specific Domains

You can leverage the `defaults` and/or `PlistBuddy` commands to:

- read/write pre-existing domains
  - VERY useful in automation and scripting
- create/read/write/delete your own domains on-the-fly
  - Even your scripts can have saved preferences!

MCX

- The “old way” of managing prefs, but still useful
  - old clients don’t speak configuration profiles

Summary

Configuration Profiles

- The New! Improved! cross-Mac OS X and iOS way of configuring user(groups) and device(groups)
- A newish technology
  - not as well documented yet as it probably will be
    - Current docs are all online, no separate PDF

acmefoo!

A co-op model of developers and trainers

- Command Line
- Sysadmin Automation
- Cocoa (Mac OS X and iOS)
- Mac OS X and iOS Deployment
- Publishing Automation
- Life Automation
- Work Automation
- Pro Photo Automation
- Pro Video Automation
- Pro Audio Automation
- Home/Business Automation
- Multimedia Automation
- ... (programming, automation, etc.)

acmefoo!

Please send email to info@acmefoo.org for:

- Ideas on Automations you'd like to see created
- Courseware you are interested in
  - attending
  - delivering
  - developing
- The Automation Mindset book
  - Designed specifically for non-programmers
    - Be able to read and use every page of MacTech (and others)
  - Winter 2011-2012

Automating OS X and iOS Configuration

acmefoo!

Scott M. Neal
smn.MG@acmefoo.org

MacTech 2011