

# Will O'Neal

Will has been around small business IT for his 23 years in the "real world" career. He's been dealing with Macs for more than 20 years, and he worked for a couple of other small Mac shops before starting Mid-Atlantic Computer Solutions in late 2002. Many of his customers date back to the late 90's as they have followed him from place to place.



MAKE ME A SANDWICH.

|

SUDO MAKE ME  
A SANDWICH.

|



WHAT? MAKE  
IT YOURSELF.

|

OKAY.

|



# Thanks to Take Control Books!

- This presentation would not have been possible without Take Control of the Mac Command Line with Terminal, published by TidBITS Publishing, inc.

# Introduction

- Before there was a GIU, there was a computer with no mouse. A display and keyboard only.
- This keyboard was the way you interacted with the computer. There was no other way
- All input was keyed in, all results were presented on the screen, or printed out
- All this changed with the Mac, which popularized the GUI back in 1984

# Basic Concepts

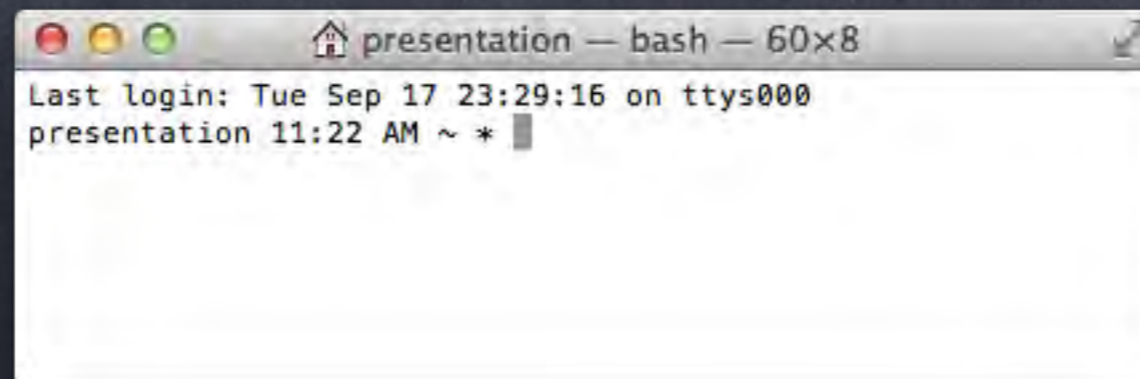
- What is Unix
- What's a Command Line?
- What's a Shell?
- What's Terminal?
- What Are Commands, Arguments, and Flags?

# What is Unix

- The OS that powers your Mac
- Has many “branches” – versions
- Each is distinct in it’s own way
- If you strip away the gui & programming interfaces (Cocoa, Carbon, & Java) you would be running Darwin, a highly customized version of Unix, available to anyone (including non-mac users) as Open Source

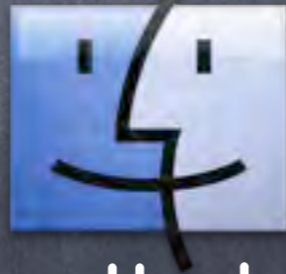
# What's a Command Line?

- A way of giving a computer a command and getting a result
- Most of the time, all input and output remains on the screen
- Only one line – the one with the blinking cursor – is the Command Prompt – the one where commands appear as you type them



# What's a Shell?

- A program that enables you to interact with a computer
- The Finder is a type of shell, but it's a GUI shell; today we will talk about Terminal.app, or any program that will create a command-line interface



- The default shell is called Bash, but there are others; Bash is where we'll spend time today.

# What's Terminal?

- It's the program that simulates a terminal, which is what you used to interact with a computer
- A terminal consisted of a display, a keyboard, and a connection to the mainframe
- Modern Terminal applications provide a terminal like connection to a shell running either on the same computer or over a network



# What Are Commands, Arguments, and Flags?

- Things you type into a terminal
- That's pretty much all you can do in a shell. There are no icons to double click on, no files to drag, no pictures to preview.

# Commands

- Verbs
- When you enter a command, you tell the computer to do something
- Very often, a single word is enough to get something done
- Example: WALK



# Arguments

- nouns
- The object of the command
- What you want the command to act on
- Can have multiple arguments
- Order can be critical
- Example: Walk to the LEFT



# Flags

- adverbs
- describes how to do something
- walk to the left QUICKLY and QUIETLY
- sometimes flags are required



# Get to Know Terminal

- Learn the Basics
- Modify the Window
- Open Multiple Sessions
- Change the Window's Attributes
- Set a Default Shell

# Learn the Basics

- When you launch Terminal you'll see an (almost)empty window that shows a command line interface generated by a shell
- Everything here is done with the keyboard
- It shows the date and time of your last login, as well as how you last logged in. You will also see the host name of the machine. The \$ signifies you are an ordinary user, as opposed to root, which is signified by #.

# Modify the Window

- This window resizes just like any other window on your computer.
- It is a good idea to resize the window because some commands generate a lot of output
- Don't full screen the window, leave enough room so you can see windows from other Apps
- Note that changes made here aren't saved between Terminal sessions unless you select Shell > Use Settings as Default.

# Open Multiple Sessions

- You can have as many windows (sessions) open as you want; you can also have multiple tabs
- The multiple sessions do not interact with each other – they are completely separate – even if you are viewing the same set of files or same program
- This is helpful because you can view the man page in one window and the program in the other

# Change the Window's Attributes

- In addition to window size, you can change window color, text color, cursor color, cursor styles, etc.
- See Terminal > Preferences > Settings
- Once you find one you like, set it as your default by going to Shell > Use Settings as Default

# Set a Default Shell

- The default shell in versions of OS X starting with 10.3 is bash
- If you started with OS X 10.0, and upgraded, your shell may be set to something different
- Check the shell you are running by entering `echo $0`
- Change your default shell in System Preferences > Users & Groups > control-click your username > Advanced Options

# Look Around

- Discover Where You Are
- See What's Here
- Repeat a Command
- Cancel a Command
- Move into Another Directory
- Jump Home
- Understand How Paths Work
- Understand Mac OS X's File System
- Use Tab Completion
- Find a File
- View a Text File
- Get Help
- Clear the Screen
- End a Shell Session

# Discover Where You Are

- `pwd`
- stands for “print working directory,”
- returns the complete path to the directory you’re currently using.

# See What's Here

- ls
- stands for "list"
- gives you the list of files in the current folder
- optional flag
  - -l = long format
  - -a = list all, including hidden
  - -h = human readable (converts bytes to mbytes)

# Repeat a Command

- up arrow, or !!
- using !!, you can add flags to the previous command
- example:  
    ls -l  
    !! -h (net result would have been ls -l -h)
- not very handy for short commands, but great for long commands
- very hand when you forget to add sudo – sudo !!

# Cancel a Command

- Control-c

or

- Command - . (period)
- command history doesn't show cancelled commands

# Move into Another Directory

- `cd`
- stands for Change Directory
- must use full path unless new location is a subdirectory (you can't `cd` to Library when you are in Desktop without doing indicating where Library is
- example: `cd /Users/bob/Library` if you are in `/Users/bob/Desktop`
- `cd ..` = up one level
- `cd ../..` = up two levels
- `cd -` = return to the previous directory

# Jump Home

- `cd ~` (tilde) = current users home directory

or

- just `cd`
- `cd ~/Library` = current users Library folder



# Understand How Paths Work

- how do you get to "My Folder" (note the space in the directory name)?
- `cd "My Folder"` – enclose the path in quotes
- `cd My\ Folder` – place a backslash before the space
- multi character wildcard – use an `*` if there is only one path, as in `cd /App*` – the `*` replaces zero or more characters
- single character wildcards – use a `?` to replace a single character, as in `ls -al 00??.jpeg`

# Understand Mac OS X's File System

- there are numerous files in the file system with the same name, or very similar names
- example – every user on the machine has a ~/Desktop folder, and the computer, and the users have /Library folders
- ., /, \, ~, and <space> have great significance



# Use Tab Completion

- tab completion allows you to start typing a file or directory name, and press tab
- if there is more than one possibility, the computer will beep; press tab again and it will show you all the possible files or directories that match
- tab completion is case sensitive!

# Find a File

- find: give it a name (or partial name) and where to start looking
- example: `find ~ -name "*microsoft*"` will look for files in the home directory with microsoft in the name
- this method is very slow because there is no index; to search quicker, narrow the location
- `find . -name "*microsoft*"` – the `.` says to search the current and all sub-directories; `/` says to search the entire disk

# Locate a File

- locate relies on a database of file and path names
- the database is only updated once a week, but once updated, is very fast
- on first use, you may have to build the database with `/usr/libexec/locate.updatedb`
- only searches files you own
- to search the entire system, update the db with `sudo`; bypasses user & system permissions

# View a Text File

- `less (or more) <filename>` to read a text file, shows page by page
- `cat <filename>` does the same thing, but shows the entire file
- `tail <filename>` displays the end of a file. Very useful for log files
- example: `tail -n 50 <filename>` shows the last 50 lines of the file
- adding `-f` forces the tail to not stop

# Get Help

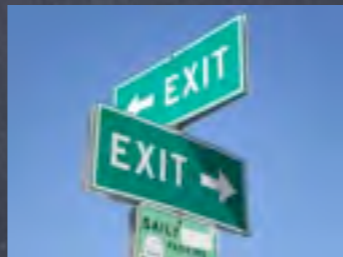
- `man` is the quickest way to get information about commands and how to use them. Sometimes the information isn't clear, but it's the best place to start
- run `man` in a new window, so you can reference the documentation and the active window side-by-side
- google is a great resource, because when the `man` pages aren't clear, someone has written about it

# Clear the Screen

- as you fill your window with commands, the window may become cluttered
- clear
- moves the command line to the top of the screen with empty space below it; you can still scroll up, though

# End a Shell Session

- `exit` will properly end a shell session
- by default, the Terminal window remains open after exiting; you can change this behavior in Terminal > Preferences



# Work with Files and Directories

- Create a File
- Create a Directory
- Copy a File or Directory
- Move or Rename a File or Directory
- Delete a File
- Delete a Directory

# Create a File

- touch <new filename> is a very fast way to create an empty file of no type
- some programs behave differently based on the existence of a file; it will also let you create files for testing so you don't destroy your computer
- touch <existing filename> updates an existing file's modification date

# Create a Directory

- mkdir = make directory
- you can use the full path to make a new directory in a different location



# Copy a File or Directory

- `cp` = copy
- requires two arguments, a source and a destination
- the first is the source file – what you want to copy
- second is the destination file – what you want to call the copy, and possibly, where to put it
- is NOT SMART – you can copy a file over a folder, or another document with no warning!
- fix this problem with the `-i` flag
- example: `cp -i ~/Desktop/file1 ~/Documents/file1`
- copy directories with the `-r` (recursive) flag
- avoid using a `/` on the end of the source directory; that copies the contents, not the directory; tab completion adds the `/` automatically!

# Move or Rename a File or Directory

- to move a directory, use mv
- to rename a directory, use mv!
- two arguments – file you want to move, and the destination directory and possibly, if you want to rename it, the destination file name.
- avoid overwriting existing files with the -i flag
- move multiple files at once; the final argument is the destination
- example: mv file1 file2 file3 ~/Documents or mv \*.jpg ~/Pictures
- you cannot use mv to rename a batch of files as in mv \*.jpg \*.JPG \*.jpeg

# Delete a File

- `rm` = remove
- `rm <filename>`
- remove multiple files with `rm file1 file2 file3`
- use wildcards with `rm *.docx` or `Microsoft*`



# Delete a Directory

- `rmdir` = remove directory
- can use multiple targets, as in `rmdir directory1 directory2 directory3`
- can use wildcards – `rmdir ~/D*`
- only works on empty directories
- to override this use `rm -r directory`



# Work with Programs

- Learn Command-Line Program Basics
- Run a Program or Script
- Run a Program in the Background
- See What Programs Are Running
- Stop a Program
- Edit a Text File
- Create Your Own Shell Script

# Learn Command-Line Program Basics

- most programs do one thing and then stop working; example ls. It lists the content of a directory, then stops running
- some programs provide no output unless there is an error – as in cp, mv and rm
- some programs are interactive, like passwd. You start the application, it asks for information, after you respond, it asks for more information, then confirmation, then it quits
- other programs like ssh and ftp work this way

# Run a Program

- running a program involves typing it's name and pressing return
- some programs live in unusual places and you need to run them using the full path; if the path of the application is not included in your PATH (capitalized), then you have to include the full path. You can modify your PATH (covered later)
- you can always run a program by typing it's full path, or by changing to it's directory and using ./ in front of the command

# Run a Script

- scripts are used to automate or simplify repetitive activities
- similar to Applescript
- usually called a shell script, it is a series of instructions run by the shell itself
- when you run the script, the shell executes the programs in the script one after another
- usually have an extension of .sh and are human readable

# Run a Program in the Background

- most of the time when you run a program, it takes over the shell, and whether it takes a second or an hour, it doesn't release the window until it has completed
- background programs let you do other tasks in the same Terminal window, and they can keep going after you quit Terminal.
- to run a program in the background, simply put a space and an & after the program name and flags or arguments
- next time you run the program, it will result in a process number and tell you that it's done along with the job name

# See What Programs Are Running

- top – similar to Activity Monitor; results update dynamically
- lets you see how many and what programs are running right this minute
- includes all programs, not just the ones controlled by the user
- flag -n with a number will limit the list to a n number of items to show
- flag -o (order) and cpu to show processes in order of CPU usage, or rsize to list processes in order of RAM they use
- example: top -n 20 -o cpu to list only the top 20 processes by CPU usage
- ps is a static listing of processes running at the moment. Only shows processes running in terminals, not all applications, unless you add flags
- you can get processes running from the Applications folder using ps -ax | grep /Applications
- that long | character is called a pipe, and it directs the output from one program to another. the pipe command is way to complex for this presentation, but you can google it!

# Stop a Program

- you can stop a program that doesn't terminate automatically a few ways
- control-c is the polite way of asking for the program to stop
- kill along with the process ID will ask the program to terminate politely, just as if you sent it a control-c
- example `kill 1234`
- where does the process ID come from? Were you listing in the previous slide? Hint: `ps`
- can also use the name of the process using a variant of kill called `killall`, as in `killall Dock`
- you can still kill a program that won't respond to control-c or kill by adding the flag `-9`, as in `kill -9 1234`; the `-9` flag says to use brute force to stop the program
- if that doesn't work, reboot! (and sometimes you will have to!)

# Edit a Text File

- vi – vee-eye for Unix geeks
- it is way to difficult to cover here, because the user interface was written by a geeks geeks geek
- I use pico or nano, which are actually the same, but they are the closest thing to a gui you'll get in the terminal
- nano "my file" will create "my file" if it doesn't exist, or open "my file" for editing if it exists, in an editor window that is complete with "menus" on the screen

# Create Your Own Shell Script

- start with an empty text file
- insert the Shebang (which tells the computer what shell to execute the script in)

```
#!/bin/bash
```

- add one or more commands

```
echo "Hello! The current date and time is:"
```

```
date
```

```
echo "And the current directory is:"
```

```
pwd
```

- close and save the file: script.sh
- to run a script, it has to have permission to run, so enter `chmod u+x script.sh`
- run the script by typing `./script.sh`

# Customize Your Profile

- How Profiles Work
- Edit `.bash_profile`
- Create Aliases
- Modify Your PATH
- Change Your Prompt

# How Profiles Work

- a profile is a file your shell reads every time you start a new session
- contains a variety of preferences for how you want the shell to work
- can be way more complicated than this topic allows time for, but we'll cover some basics

# Edit .bash\_profile

- now that you've googled bash profiles, how to you get one on your Mac?
- `nano ~/.bash_profile`
- if the file exists, nano will open it for editing; if not, nano creates it for editing
- to load the profile after editing it use

`source .bash_profile`

# Create Aliases

- do you ever switch back and forth between Unix and DOS, but can't remember how to get a directory listing on the system you are using right now? An alias would let you enter `dir` and have it run `ls`!
- you can also set up a shortcut for longer commands, such as `alias mvpics="mv *.jpg ~/Pictures/"`
- good examples: `alias ls="ls -alh"; alias copy="cp -i"; alias move="mv -i"`

# Modify Your PATH

- when you run a program, the shell looks for it in predetermined places
- when you start scripting, you may want to specify additional locations where programs or scripts are located
- if you store your own scripts in ~/Documents/scripts, you should add that directory to your PATH by putting this in your profile
- example: `export PATH=$PATH:~/Documents/scripts`

# Change Your Prompt

- ever wonder how or why your computer tells you what the name of the machine is before every command?
- example: Wills-MacBook-Air: ~wo\$
- change that by adding `PS1="I love bash! "` to your profile. Include the space!
- prompts can include variables: `\u`: to include your shortname; `\h`: for your computers name; `\w`: for the current directory; `\d`: for the date; `\@`: for the time
- example: `PS1="\u \@ \w [$PWD] * "`

# Real World Uses

- Get the Path of a File or Folder
- Open the Current Directory in the Finder
- Open a Hidden Directory Without Using Terminal
- Open the Current Folder in Terminal
- Open a Mac OS X Application
- Open a File in Mac OS X

# Get the Path of a File or Folder

- If you don't know the path of the target file, or don't want to type it in, you can, in the Terminal window, type the command, followed by a space, and then drag the file or folder to the window
- when you release the mouse button, Terminal copies the path of the file or folder you dragged onto the command line

# Open the Current Directory in the Finder

- you may wish to see, graphically, wherever you are in the Finder
- try this: open .

# Open a Hidden Directory Without Using Terminal

- If all you want to do is open a window to some normally hidden location, you can use the Finder command Go > Go to Folder
- enter the whole path (tab completion works!) and hit enter!

# Open the Current Folder in Terminal

- what if you are looking in a folder and want to open a terminal window to that location?
- download this <http://code.google.com/p/cdto/>
- drag the downloaded file to any Finder windows toolbar, and you'll have quick access to the directory in Terminal from the Finder

# Open a Mac OS X Application

- `open -a Safari`
- no need to tell it a location; it's smart and will search for the application anywhere on the disk

# Open a File in Mac OS X

- open picture1.jpg
- will open that file with the default application
- open -a /Adobe\ Photoshop picture1.jpg
- will open that file using Adobe Photoshop

# Log In to Another Computer

- Start an SSH Session
- Run Commands on Another Computer
- End an SSH Session

# Start an SSH Session

- `ssh user-name@remote-address`
- if this is the first time you've connected to a particular remote machine, you will see something like:

```
The authenticity of host 'macbook-pro.local (fe80::20c: 74ee:edb2:61ae%en0)' can't be established.
```

```
RSA key fingerprint is d0:15:73:75:04:9a:c3:2d: 5b:b1:f8:c0:7d:83:52:ef.
```

```
Are you sure you want to continue connecting (yes/no)?
```

- assuming you are comfortable proceeding, type `yes` and press return, and you will see:

```
Warning: Permanently added 'macbook-pro.local., fe80::20c:74ee:edb2:61ae%en0' (RSA) to the list of known hosts.
```

- following that is a password prompt. You know what to do!

# Run Commands on Another Computer

- once you are logged in, you can run commands exactly as you do on your local computer
- note: your default shell might not be the same
- note: your `.bash_profile` doesn't apply to the remote computer
- any files you interact with will only happen on the remote computer. If you "open" a file, it will open on the remote machine, not your local machine!

# End an SSH Session

- simply type `exit`
- it's a good idea to do this every time, because it will shut down any processes you started in that session. It might not matter locally, but if you leave something running on someone else's computer, it might have unintended consequences

# Digging Deeper

- Understand Permission Basics
- Change an Item's Permissions
- Change an Item's Owner or Group
- Perform Actions as the Root User

# Understand Permission Basics

- permissions will control what you can do and what you can see in the terminal. If you've seen the locks on folders in the Finder, they are the same in the Terminal
- the read permission will allow the user to open and see what is in a file (signified by r)
- the write permission means one can modify or delete an item (signified by w)
- the execute permission means that it can be run if it's a program or a script; if it's a directory, it means someone can list it's contents (signified by x)

# Users, Groups, and Everyone Else

- User: In terms of file permissions, the term user means the owner of a file or directory. (signified by u)
- Group: Each file and directory also has an associated group—one or more users for whom a set of permissions can be specified (signified by g)
- Others: Every user who is neither the owner nor in the file's group is lumped into the "others" category (signified by o)

# Change an Item's Permissions

- to grant group write access to file1, enter `chmod g+w file1`
- to remove other's execute permission to file1, `chmod o-x file1`
- you can affect multiple users at once: to add read access for user, group, and other, enter `chmod ugo+r file1`
- if you aren't the owner of the file, you must `sudo`

# Change an Item's Owner or Group

- to change an item's owner, group, or both, use `chown` (change owner)
- if you are changing both the owner and the group, separate them with a `:`
- example: `chown bob file1` to change the owner of `file1` to user `bob`
- example: `chown bob:accounting` to change the owner and the group to user `bob` and group `accounting`
- to change only the group, `chown :accounting file1`
- all of these will fail without the next section!

# Perform Actions as the Root User

- Mac OS X, and all Unix and Unix like operating systems prevent users from viewing or altering files that don't belong to them
- This is why you can't see or access anything in the System folder!
- There is a hidden account, called root, and that user has virtually unlimited power to screw up anything on the computer
- An admin user can temporarily assume the capabilities and authority of the root user with sudo – which stands for “superuser do”
- type in sudo command and you will be prompted to enter your password prior to the command executing
- if you are a security conscious user and your primary account is not an admin account, you'll need to switch users prior to using sudo by running su user-name first

# Command-Line Recipes

- Change Defaults
- Perform Administrative Actions
- Modify Files
- Work with Information on the Web
- Manage Network Activities
- Work with Remote Macs
- Troubleshoot and Repair Problems
- Get Help in Style
- Do Other Random Tricks

# Change Defaults

- Show Hidden Files in the Finder

```
defaults write com.apple.finder AppleShowAllFiles TRUE;  
killall Finder
```

- Prevent Dock Icons from Bouncing

```
defaults write com.apple.dock no-bouncing -bool TRUE;  
killall Dock
```

- Deactivate Dashboard

```
defaults write com.apple.dashboard mcx-disabled -boolean  
YES; killall Dock
```

- Expand Save Dialogs by Default

```
defaults write -g NSNavPanelExpandedStateForSaveMode -bool  
TRUE
```

# Perform Administrative Actions

- Use Software Update from the Command Line

```
sudo softwareupdate -i -a
```

- Find Interesting Stuff in Log Files

```
grep error /var/log/system.log
```

- look for all entries involving Time Machine

```
grep backupd /var/log/system.log
```

# Modify Files

- Change the Extension on All Files in a Folder
- This requires you writing a shell script!

```
#!/bin/bash
for f in $3/*. $1; do
    base=`basename $f . $1`
    mv $f $3/$base. $2
done
```

- put the file in your PATH and that it is executable
- to run it, enter the script name followed by the old extension, the new extension, and the directory in which you wish to make the change
- example: rename.sh JPG jpeg ~/Documents

# Modify Files part 2

- Decompress Files

```
tar -xf archive.tar
```

```
tar -xzf archive.tar.gz
```

```
tar -xjf archive.tar.bz2
```

- Convert Documents to Other Formats

```
textutil -convert doc file1.rtf -output file2.doc (this actually  
converts and saves the file with a different name!)
```

- Convert a File from Word (.doc) to HTML

```
textutil -convert html file1.doc
```

# Work with Information on the Web

- Download a File

```
curl -s -S -O URL
```

- Save a Local Copy of a Web Page

```
curl URL > filename.html
```

- Put the Source of a Web Page on the Clipboard

```
curl URL | pbcopy
```

# Manage Network Activities

- Get Your Mac's Public IP Address

```
curl -s http://www.showmyip.com/simple/; echo
```

- Get a List of Nearby Wi-Fi Networks

```
/System/Library/PrivateFrameworks/Apple80211.framework/  
Versions/A/Resources/airport -s
```

- To see a list of all processes accessing the Internet, enter:

```
sudo lsof -i
```

# Work with Remote Macs

- Use Secure Screen Sharing via SSH

put the following in your `.bash_profile` file and then start a new shell session:

```
alias stss="(sleep 15; open vnc://127.0.0.1:5901) & ssh  
-C -4 -L 5901:127.0.0.1:5900"
```

- to start a session

```
stss user-name@domain.com
```

- enter your password for the remote computer,  
then authenticate a second time

# Troubleshoot and Repair Problems

- Delete Stubborn Items from the Trash

```
sudo rm -ri ~/.Trash/*
```

- If that doesn't work, try each of these until the Trash is empty:

```
sudo rm -ri /.Trashes/*
```

```
sudo rm -ri /Volumes/*/Trashes/*
```

- Figure Out Why You Can't Unmount a Volume

```
lsdf | grep /Volumes/VolumeName
```

# Get Help in Style

- Read all man Pages in Preview

put the following lines in your `.bash_profile`:

```
psman()  
{  
man -t "${1}" | open -f -a /Applications/Preview.app/  
}
```

- to view a man page in Preview

`psman command`

# Do Other Random Tricks

- Take a Screenshot

`screencapture ~/myscreen.png`

- Use Text-to-Speech from the Command Line

`say "Hello there"`

`sleep 60; say "One minute has elapsed"`

- Find a File by Content

`grep -R "your text" .`

This searches from the current directory down. If you want to time consumingly search the entire disk, replace the `.` with a `/`, or whatever directory you want to search. This search finds partial matches without using wildcards.

# Contact

Will O'Neal

Mid-Atlantic Computer Solutions

[woneal@4macsolutions.com](mailto:woneal@4macsolutions.com)

703.236.5800 x 101