Introduction to the Command-Line Mindset (distilled version...)

acmeFoo

Scott M. Neal
smn.mg@acmefoo.org
MacTech Seattle Mar 2013

Command Line Mindset: Agenda

Goal:

- To see the mindset of the Command Line Interface
  - (and never worry about memorizing commands again...)

Materials derived from 2-day acmeFoo CLI101 course

Why use Command Line?

Command Grammar and Syntax

Substituting User

Shortcuts and Special Characters

File Viewing Commands

Documentation, Searching, & Editing

Commands, Commands, Commands...

Installing Open Source SW

Why use Command Line?

Why learn about the Unix Command Line?

Remote administration
- Slow connection, or GUI not available remotely

Single-user mode

Recovery HD

Security and monitoring

Many actions only available at command line

Running a command as a different user

Combine commands with a pipe |

Automation and Scripting

Troubleshooting

Several “Apple Only” command line tools

Open Source tools which are only on CLI

Deployment Scripting

Media Management

The Command-Line Mindset

The Command-line does not utilize the same mindset as most GUI programs

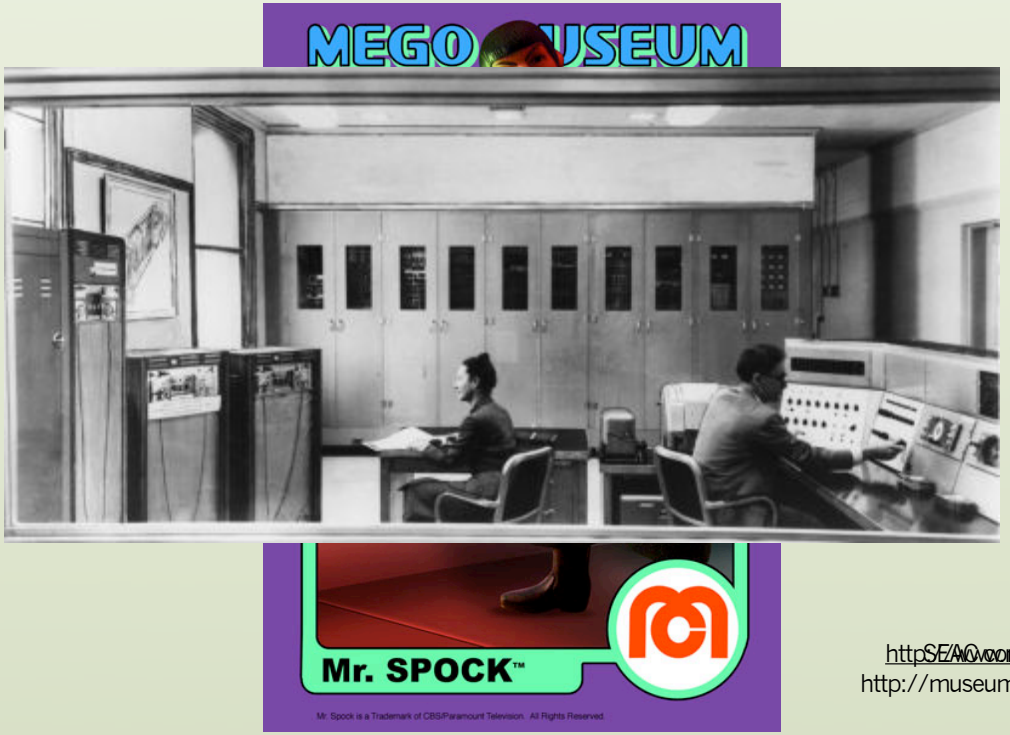
- much more “picky”

There are particular “rules” that once learned, make it much easier to understand what is going on without memorization

- Many introductions to the command-line focus on memorizing commands
  - You don’t need to memorize much, surprisingly
- Note that the name of this presentation is “The Command Line Mindset”, not “Here’s a bunch of things to memorize”

Brief History of the Command Line Interface

Remember that the original interface to modern digital computers was toggle switches, patch cables, and lights



http://museum.nist.gov/exhibits/seac/SEACOVER.HTM

MACTECH

Brief History of the Command Line Interface

When the Command Line Interface (CLI) was developed, it was a major breakthrough

http://www.columbia.edu/cu/computinghistory/teletype.html

The Unix Command Line: a Modern Perspective

Since the Unix Command-Line was originally developed in the 70s, is it still useful?

The Unix Shell

Provides the primary interactive and scripting interface on Unix systems

- What most people call “Unix” is really a **Unix shell**, or often a **Command-Line Interface (CLI)**

A Unix Shell is an application program

- Can be used **interactively**
  - Will have a **prompt**
- Can be used as a Shell **script interpreter**
  - Unix shells are not the only scripting environments…

Shells are independent of the underlying operating system

- Different flavors of Unix
- Non-Unix OSes
  - Unix shells have been ported to Windows and other operating systems

Unix shells (cont.)

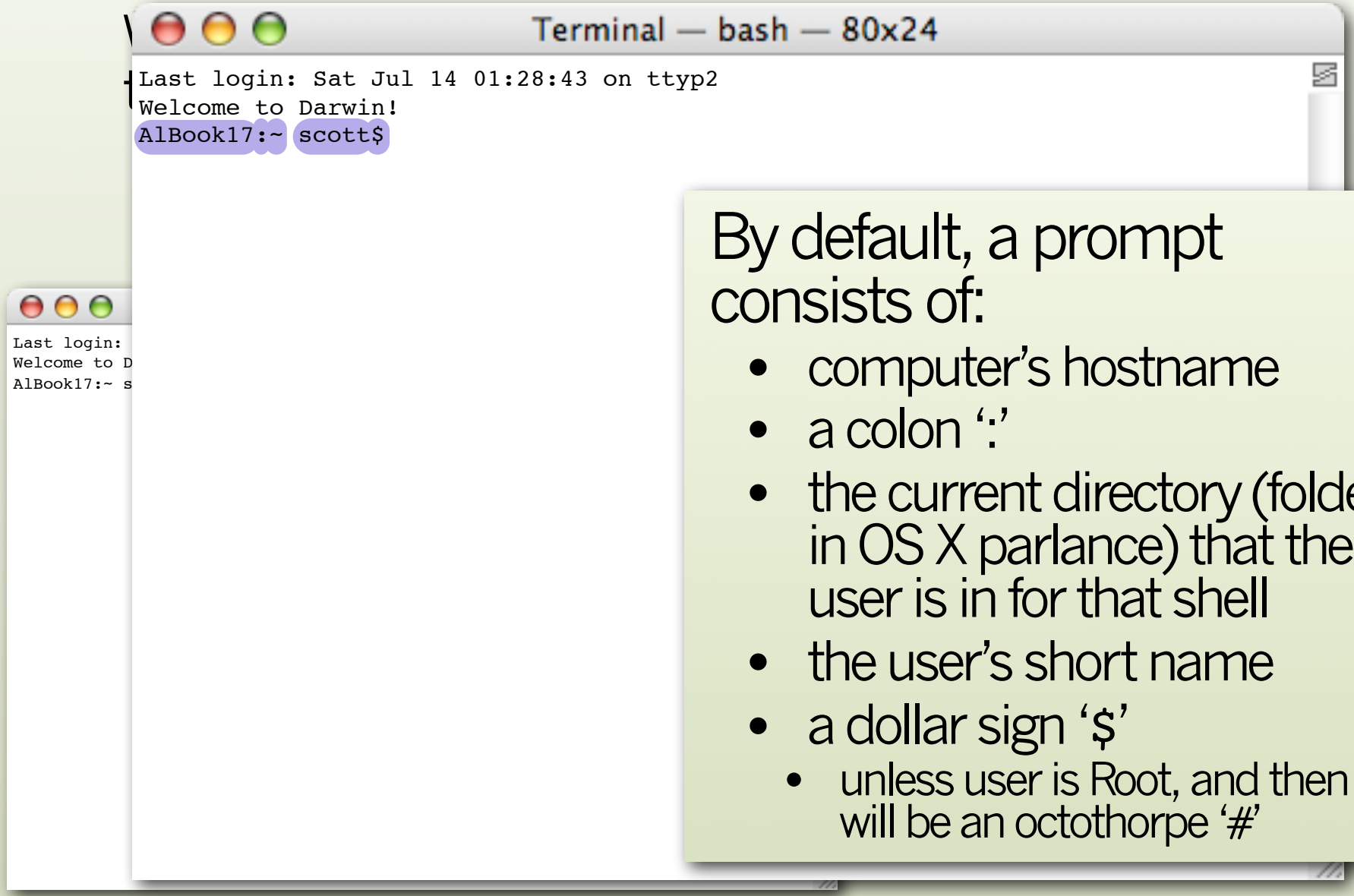
Typical Unix shells:

- `bash`
  - Default shell on Mac OS X > 10.2
  - Most popular on Linux
- `tcsh`
  - Default shell on Mac OS X <= 10.2
- `sh`
  - First common Unix shell, still used for system scripts
- `zsh`
- `ksh`
- ... (there are MANY more)

An entry in Directory Services provides default shell (or none) for each user

Scripting languages like Perl, Python, etc. are NOT Unix shells

Interactive Shell Prompt



By default, a prompt consists of:

- computer's hostname
- a colon ':'
- the current directory (folder in OS X parlance) that the user is in for that shell
- the user's short name
- a dollar sign '$'
  - unless user is Root, and then it will be an octothorpe '#'

Interactive Shell Prompt

To save space in this guide, we will often use '$' to represent the entire Shell prompt, so

```
AlBook17:~ scott$
```

will be abbreviated as

```
$
```

Terminal

A Mac OS X GUI application which provides one (or many) Unix shell(s) on the local machine

- Terminal itself is NOT a Unix shell

Terminal.app is Available:

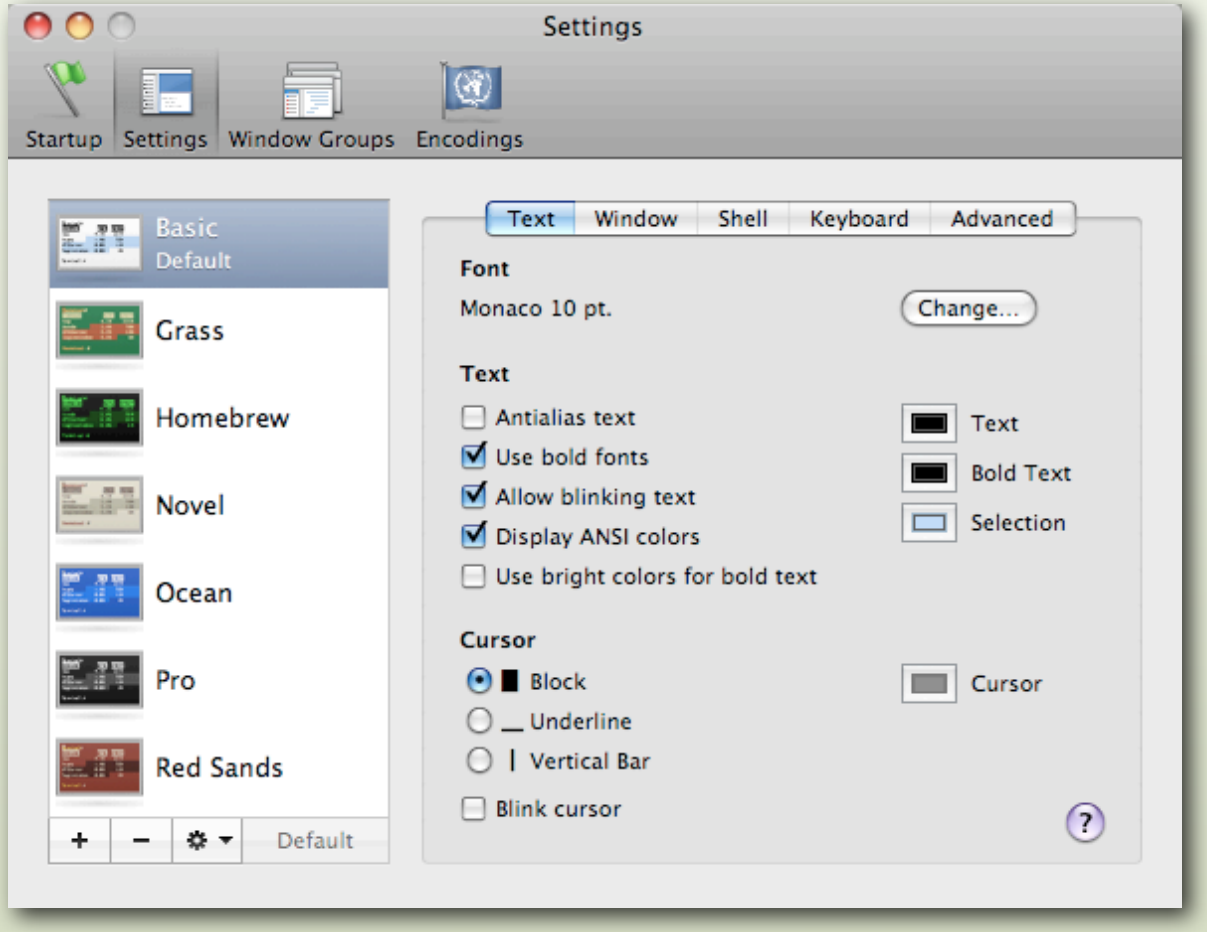
- While logged into the GUI
- When booted from installer CD/DVD or Recovery HD
  - Mac OS X 10.3 and later

Highly configurable and customizable

- Color of shell window, text, font
  - Should use only monospace fonts like Monaco or Courier
- Transparency of shell window
- etc.

Terminal Preferences

Investigate the Terminal settings and preferences to customize your command-line experience



Command Line Grammar and Syntax

Unix Commands

A Unix **command** is any script or program that is executable by Unix

- Each file (and folders too, but for another reason) have an e**x**ecutable bit called ‘x’ indicating executability

The name of a command is often case-sensitive

- If the command is `ls`, you shouldn’t use `Ls` or `LS` etc.
- If the command is `SetFile` you shouldn’t use `setfile` or `SETFILE` or any other variant

A Well-formed command (or commands) form(s) a **Command-Line**

- Hence the name Command-Line Interface (CLI)

Commands return error messages when unsuccesful

- Some messages are more useful than others
- Look at the feedback from the command!

Unix Commands

There are FAR too many Unix commands to memorize all of them

- built-in
- downloadable

It's much more useful to learn the "grammar" of the Unix command-line, and then look up the "words" (commands) as you need them

We will start with some basic commands

- `ls`
- `cd`
- `cat`
- `grep`
- `echo`
- `mkdir`
- `rmdir`
- `cp`
- `rm`
- `mv`
- `ssh`

... but will focus on grammar (and syntax)

The `ls` Command

The `ls` command tells the Unix shell to display a **ls**ting of files and folders (directories)

Here are some examples of `ls`:

- `ls`
- `ls -lA /`
- `ls -lA ~/Documents`
- `ls -lA ~/Documents > /tmp/doc_ls_list.txt`
- `ls -lA ~/Documents >> /tmp/doc_ls_list.txt`
- `ls -lA ~/Documents | grep -iR pdf`

"Learn by Doing"--walk through each of these with me (next pages)

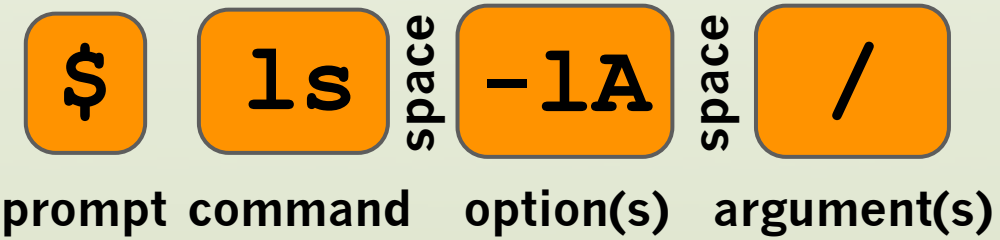
The `ls` Command: Options & Arguments

We start with just the **command** itself:

```
ls
```

We then add some **options** and **arguments** to the command to create a long listing and show hidden files on the root of the filesystem:

```
$ ls -lA /
```



Grammar:

- Commands Verbs
- Arguments Nouns
- Options Adjectives/Adverbs
- "spaces" serve the same purpose as in English

The `ls` Command: Options & Arguments

We then change the arguments to the command to create a long listing in our own Documents folder (still showing hidden files, we didn't change the options):

```
$ ls -lA ~/Documents
```

prompt **command** **option(s)** **argument(s)**

The `ls` Command: Output Redirect to a File

Not only can Unix output each command's results directly to the Unix shell (where you can see it immediately) but Unix can also **redirect output** to any file path using the > character:

```
ls -lA ~/Documents > /tmp/doc_ls_list.txt
```

- If the file already exists, it will be erased with no warning!

The `ls` Command: Output Redirect to a File

Instead of creating a new file for the output, **append** to a pre-existing file using >>:

```
ls -lA ~/Documents >> /tmp/doc_ls_list.txt
```

- This will create a brand-new file if one doesn't already exist

The `ls` Command: Output Redirect (review)

redirect output to a file:

```
$ ls -lA ~/Documents > /tmp/doc_ls_list.txt
```

append to a pre-existing file:

```
$ ls -lA ~/Documents >> /tmp/doc_ls_list.txt
```

Notice that in both cases the direction of the chevron (s) points in the direction of the flow of information

The `ls` Command: Command Piping |

Finally, instead of sending the output of `ls` to a file, we instead send it to another command called `grep`:

```
ls -lA ~/Documents | grep -iR pdf
```

- This is called **piping**, since we are connecting the output of `ls` to the input of `grep`
- the | signifies the **pipe** connection
  - Notice there are no chevrons here--that is for file redirecting
- The `grep` command **parses** its input and acts upon it
  - `grep` is used to search for **Regular Expressions** and show matches

The `cp` Command

The `cp` command **co**p**i**es files or folders (directories)

Here are some examples of `cp`:

```
$ cp ~/Documents/MyNovel /Volumes/ExternalFW/MyNovel
$ cp /Users /Volumes/ExternalFW/Backup20070629
```

The `scp` command is a combination of `ssh` and `cp`:

```
$ scp /Users remotemachine:/Storage/Backups/Backup20070629
$ scp remotemachine:/etc/openldap/ldap.conf ~/Desktop
```

Grammar and Functionality:

- Some commands, like `cp` (and unlike `ls`), do not output anything upon success
  - Only get negative feedback

The `rm` Command

The `rm` command **rem**oves files or folders (directories)

There is no "trash" in the command line--if you `rm` a file, it is GONE

Here are some examples of `rm`:

```
$ rm ~/Documents/MyNovel
$ rm -Rf ~/Library/Preferences
```

- This is probably NOT something you would want to do...

Grammar and Functionality:

- `-R` is a common option amongst CLI commands
  - There are often exceptions--some commands do not follow guidelines

The `mv` Command

The `mv` command **mov**es files or folders (directories)

- Renaming a file is basically "moving it" from one name to another, so `mv` is also used for renaming

Here are some examples of `mv`:

```
$ mv ~/Documents/MyNovel /Volumes/ExternalFW/MyNovel
$ mv ~/Documents/MyNovel2 ~/Documents/MyNovel_FirstDraft
```

Instead or `rm`'ing a file, consider `mv`'ing it instead

- In many cases, you may wish to access the file again
  - Preferences
  - Configuration files

Grammar and Functionality:

- Like most CLI commands, there is no undo!
  - But you can always `mv` the name back...

The echo Command

The echo command outputs text

Here are some examples of echo:

```
$ echo “Hello there”
Hello there
$ echo “How are you?”
How are you?
$ echo “Hello There, How are you?” > /tmp/aTextFile
$
```

ssh

The **`ssh`** program provides a **S**ecure **SH**ell to a remote computer shell account

- replaces older, insecure protocols like `telnet`, `rlogin` and `rsh`

Both authentication and communication are encrypted

- Many protocols only encrypt the authentication, or don't encrypt either

Tests authenticity of remote "server", and logs information in local file

- Ensures that "man-in-the-middle" attacks difficult to do

Using ssh

Remote Login must be enabled on "server"

- Mac OS X: Sharing pane of System Preferences

Authentication occurs on remote "server" using its Directory Service configuration

- If you want to log onto a remote machine, you must have an account there--local machine authentication doesn't count!

Authentication methods:

- Username/Password
- Public/Private key
  - Convenient--no need to enter a password when logging in
  - Necessary for scripting using ssh--secure by design

ssh in Action: Password

To log in remotely to a computer www.acmefoo.org with login name scott:

```
$ ssh scott@www.acmefoo.org
```

- or

```
$ ssh www.acmefoo.org -l scott
```

The first time you log into a host, you will see a message like this:

```
The authenticity of host 'www.acmefoo.org (86.75.30.9)' can't be
established.
RSA key fingerprint is 2d:c5:3c:f4:17:20:9a:16:9d:d8:fd:
92:96:06:cb:5d.
Are you sure you want to continue connecting (yes/no)?
```

When you say "`yes`" (fully typed out), the remote server will be added to *your* `~/.ssh/known_hosts` file

- When you reconnect to that host, `ssh` checks validity
- If a remote host becomes invalid, you will get a more severe WARNING

Arguments & Options: Review

A command is like a verb:

- `ls` a folder or file...
- `mv` or `cp` a folder or file...
- `grep` a folder or file for particular information...

Arguments are like nouns:

- the files/folders to `ls`...
- the files/folders to `mv` or `cp`...
- the files/folders to look inside of, and the information to `grep` for...

Options are like adverbs

- How should the CLI `ls` the folder or file?
- How should the CLI `mv` or `cp` the file?
- How should the CLI `grep` for information inside of files/folders?

Quick Synopsis

Unix **Commands**

- may (or may not) have **Arguments**
- may (or may not) have **Options**
  - Options may be combin-able, or not
    - may use one - or two --
- Spaces separate commands, arguments and options

Output can be **redirected** (and as will shall soon see, so can input) to files with chevron characters

- one > to create a new file
- two >> to append to a pre-existing file

The output of one command can be **piped** into another command

- no files are involved--the connection is direct

ALWAYS look for errors in command output!

- “nothing” usually means things went ok

Substituting User

Substituting User

Sometimes it's useful to spawn a Unix shell or run a command as though you are a different user than the user you are currently logged in as

Unix provides a way to spawn a shell as a **s**ubstitute **u**ser (as long as you have the correct password)

```
$ su username
```

- If you don't specify *username*, defaults to the root user
  - Which is not enabled for login by default on Mac OS X (although it is on Mac OS X Server)

Substituting User

If you only want to run a single command as someone else (usually as root), you can **sudo** (**s**ubstitute **u**ser **do**)

- On a default Mac OS X system, you must be logged in as an administrator and have the correct administrator password to run **sudo**

```
$ sudo -u username command
```

  - If you don't specify **-u** *username*, defaults to the root user

The first time you **sudo**, you will get a warning

By default, you cannot **sudo** as a non-admin user

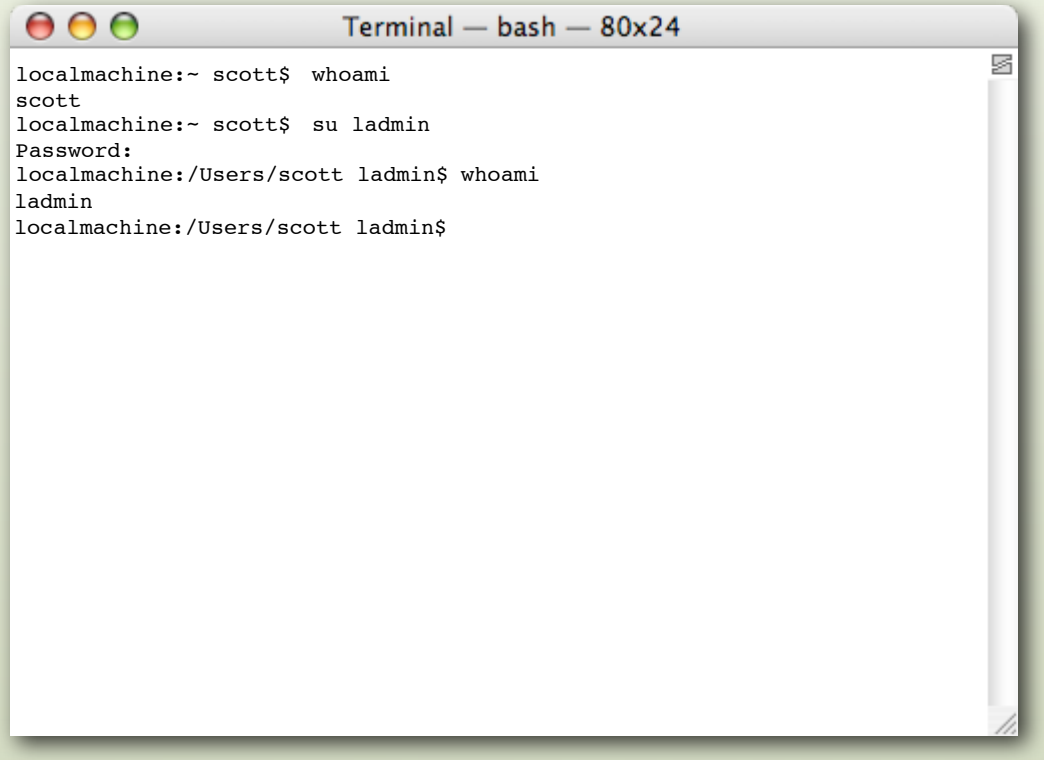
- but you can **su** to an admin user first, and *then* **sudo**

Who Am I?

When you start substituting users, you may forget which user you currently are...

There is a command to help you

```
$ whoami
```



Shortcuts and Special Characters

Unix: All That Typing! So Picky! Unfriendly!

The Unix shell is not known for its friendliness

- Very rarely do you get positive feedback, only negative...

"The most annoying thing about the CLI is all the typing, and how picky the CLI is about case-sensitivity and spelling"

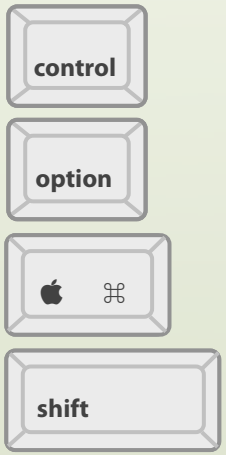
- Even Unix gurus HATE to type...

"I'm a Mac user, I'm creative, I shouldn't need to be worried about such trivialities as correct syntax"

- Let the Shell help you--it tries to help you succeed!

Modifier Keys

Don't forget your modifier keys and their Mac OS abbreviations



^ Control

⌥ Option (sometimes alt)

⌘ Command (sometimes Apple)

⇧

Some important CLI keys are not modifier keys

Escape

⇥ Tab (forward)

Shell Shortcuts

Tab completion	Shell will automatically complete the file path for you • If you haven't given it a typo! • If there is more than one choice, use double-Tab to show the multiple choices • Case-sensitive (even though file system itself usually isn't)
Left & Right Arrow keys	Move cursor without deleting any characters
Up & Down Arrow keys	Show previous lines entered into the shell

Shell Shortcuts

Delete key or `^H`	Delete character to left of cursor, move remaining characters to the left
`^A`	Move to beginning of line
`^E`	Move to **E**nd of line
`^L`	c**L**ear entire Shell screen by blanking out lines
`^U`	**U**ntype from cursor to beginning of line
`^K`	**K**lear from cursor to end of line

Shell Shortcuts

`^T`	Take character to left of cursor, **T**ranspose it with the character to the right, move other characters left
`^F`	Same as right arrow key: "**F**orward" one character at a time
`^B`	Same as left arrow key: "**B**ackward" one character at a time
`Esc F`	Move "**F**orward" one word at a time
`Esc B`	Move "**B**ackward" one word at a time

Shell Shortcuts

Return or Enter or `^M` (often loosely known as **End of Line** or **EOL**)	Send your command line to the Shell so it can (attempt to) execute it • You do NOT need to be at the end of the line to press Return/Enter/etc.

Terminal Application Shortcuts

⌘-K

- "Klear" scrollback buffer (menu shortcut)
- Note that it's ⌘-K (a Mac OS X application menu shortcut), not `^K` (which is for the shell, and does something different!)
- Also note that it's not the same as `^L`, which clears the Unix shell screen, but not the entire scrollback buffer

Drag-and-Drop path from Finder

- This will knock the socks off your LINUX buddies...

Terminal Application Shortcuts (cont.)

⌥-click (option-click) to move cursor

- Does NOT work if command-line is longer than one line in the shell

These shortcuts are specific to Terminal, and will not work in

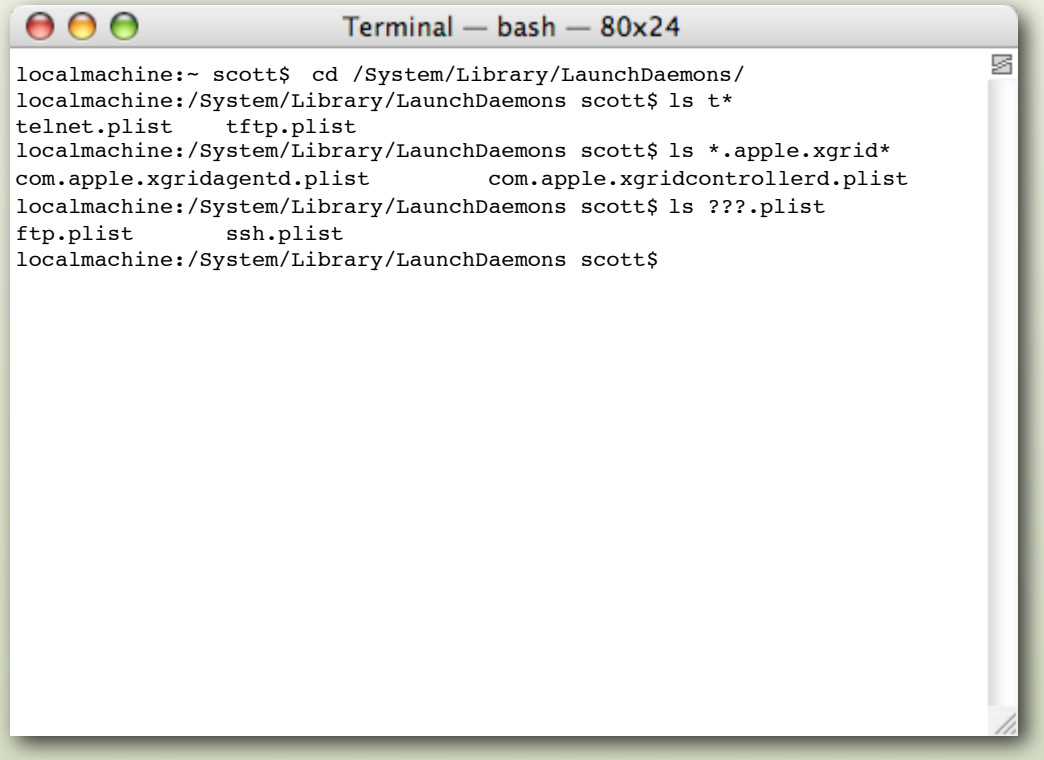
- Single-User
- X11
- Serial Port
- `>console`
- `ssh`

Wildcard characters

The asterisk '*' is a wildcard that matches anything.

The question mark '?' matches any single character.

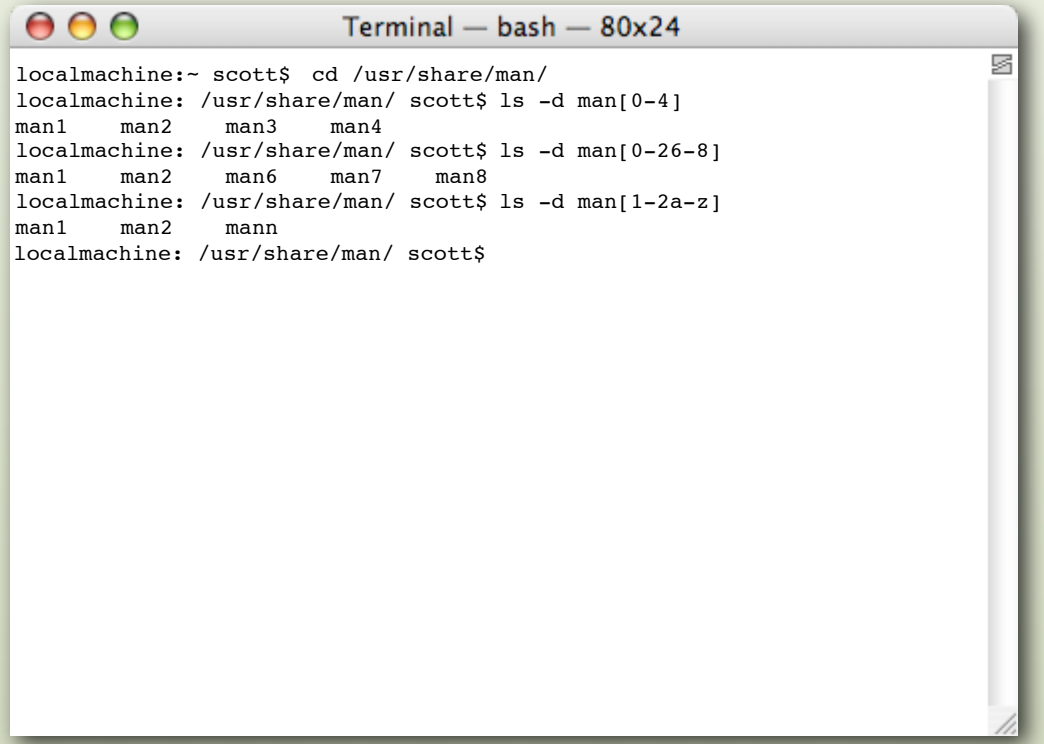
Wildcards characters are sometimes called **globbing** characters



Wildcard characters

Ranges of characters placed between brackets '[' & ']'

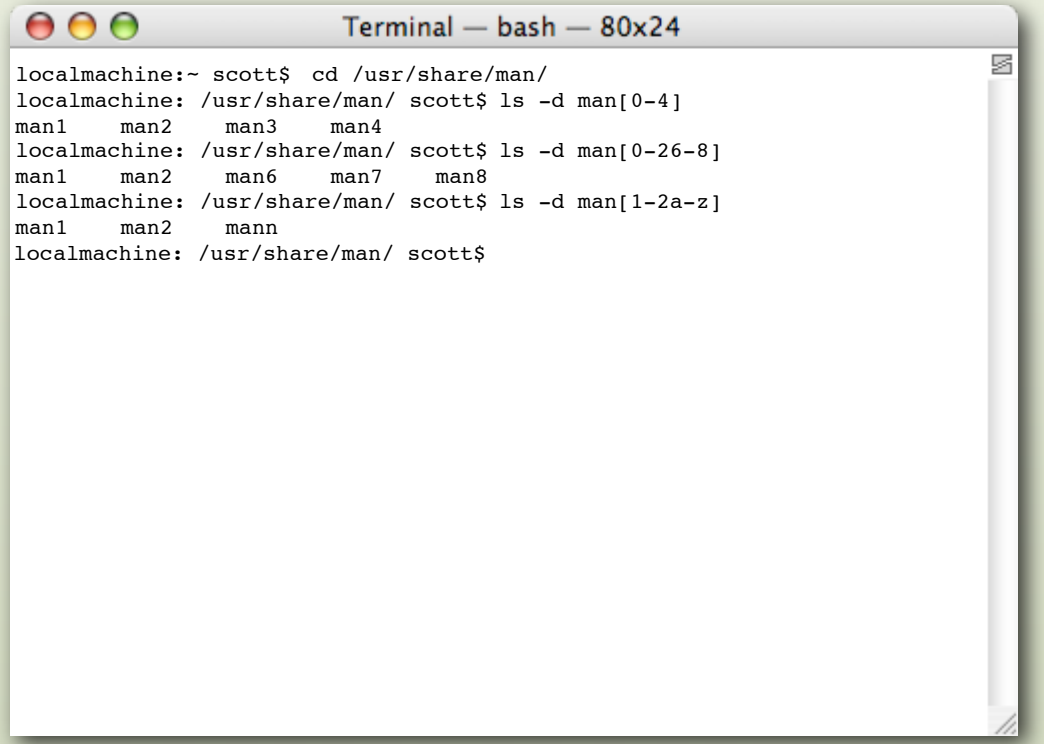
- `[0-9]` will match any single numeric digit
- `[a-z]` will match any single lowercase letter
- `[a-zA-Z0-9]` will match any single lower or uppercase letter or numeric digit



Wildcard characters

Different characters placed between braces '{' & '}'

- do NOT put spaces before or after the comma(s)
- `{jpg,jpeg}` will match either jpg or jpeg
  - as will `jp{,e}g`
- `{jpg,jpeg,JPG,JPEG}` will match either jpg, jpeg, JPG, or JPEG

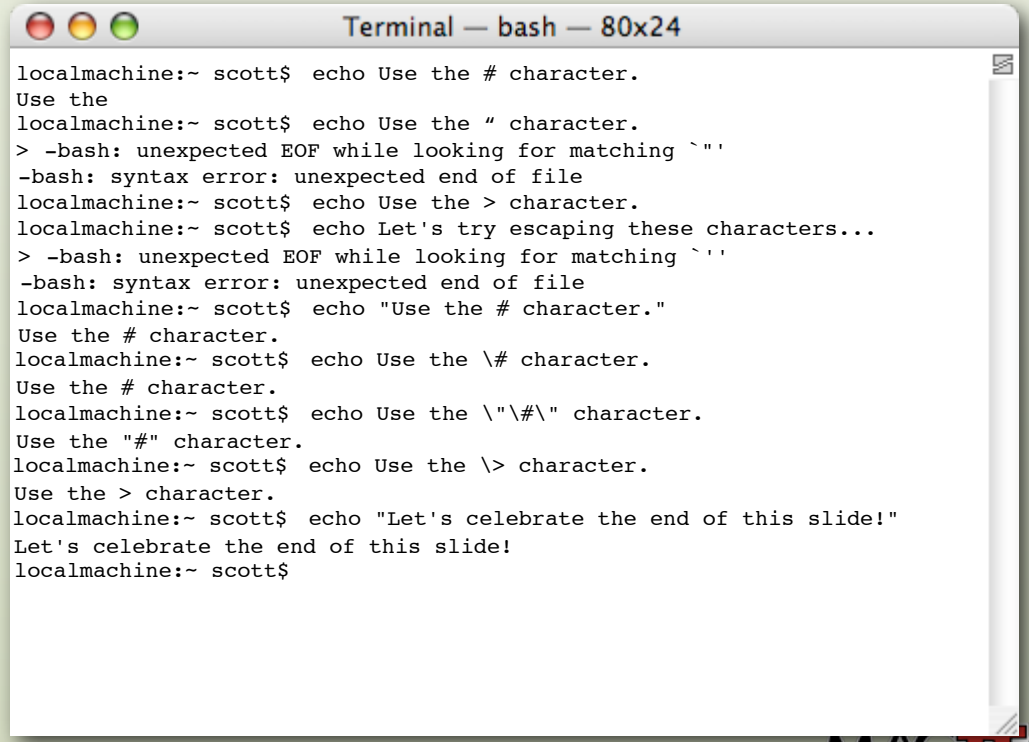


Shell Special characters

Include $ # [ ] ! = < > . , & ; | “ ‘ ` ( ) \ / ~ <space> and the globbing/wildcard characters

- The Shell wants to “act” upon them when it sees them, even if Unix (including filenames) allows them

Cannot represent themselves unless **quoted** or **escaped** with the backslash ‘\’ character.



Special characters: Space

A space is special because it is a **delimiter**

- It's the main way a shell knows how to separate commands, options, and arguments
- Spaces in file paths must be quoted or escaped with the backslash character '\' so the file path is considered to be a single entity, not multiple file paths
  - Examples:

```
$ ls "My List"
$ ls My" "List
$ ls My\ List
```

  - Unix Shell tab completion will use '\'
    - Speaking of tab completion... Trailing '/' sometimes needs to be deleted for certain commands

Line Continuation Characters

The shell parses what you type in as a single "Command Line"

- A single Command Line may be multiple "screen" lines

If you want to treat multiple lines as a single Command Line, you must use the shell's **Line Continuation** character '\'

- You are really just "escaping" your EOL

Example:

```
$ echo "wow I sure have a lot to say but I'm \⏎
> running out of room..."
```

Line Continuation Characters

Typically seen in scripts and documentation

- easier for the editor to format and for you to follow

Which is clearer about what you are supposed to do,

- this:

```
echo “wow I sure have a lot to say but I’m running out of
room here because I talk a lot...”
ls /Applications
```

- or this:

```
echo “wow I sure have a lot to say but I’m running \
out of room here because I talk a lot...”
ls /Applications
```

File Viewing Commands

File Viewing Commands

Display the contents of a text file a page at a time:

```
more file          less file
$ less /etc/named.conf
$ find / -name "*.app" | less
```

- These commands are called **pagers** because they show output one page at a time

Show the last n lines of a file (default is 10)

```
tail [-n num_of_lines] file
$ tail -20 /var/log/system.log
```

Show the tail end of a file as it's being modified live

```
tail -f file
$ tail -f /var/log/system.log
```

Show the head (beginning) of a file n lines at a time (default is 10)

```
head [-n num_of_lines] file
$ head -20 /var/log/system.log
```

File Viewing Commands (cont.)

Find a file with certain attributes:

```
$ find /Users -name \*.mp3
$ find /tmp -newer /tmp/checkfile.txt
```

Show the contents of a file without paging

```
$ cat /var/log/system.log
```

Concatenate multiple files to make one file

```
cat file1 [... filen] > bigfile
$ cat /var/log/system.log* > /tmp/bigOlLogFile
```

Append a text file to the end of another text file:

```
cat file1 >> file2
$ cat myLogFile >> /tmp/bigOlLogFile
```

Search within a file, folder, or `STDIN` for a regular expression:

```
$ grep -iR “scott” /tmp
```

Documentation, Searching, & Editing

Searching for Info

man

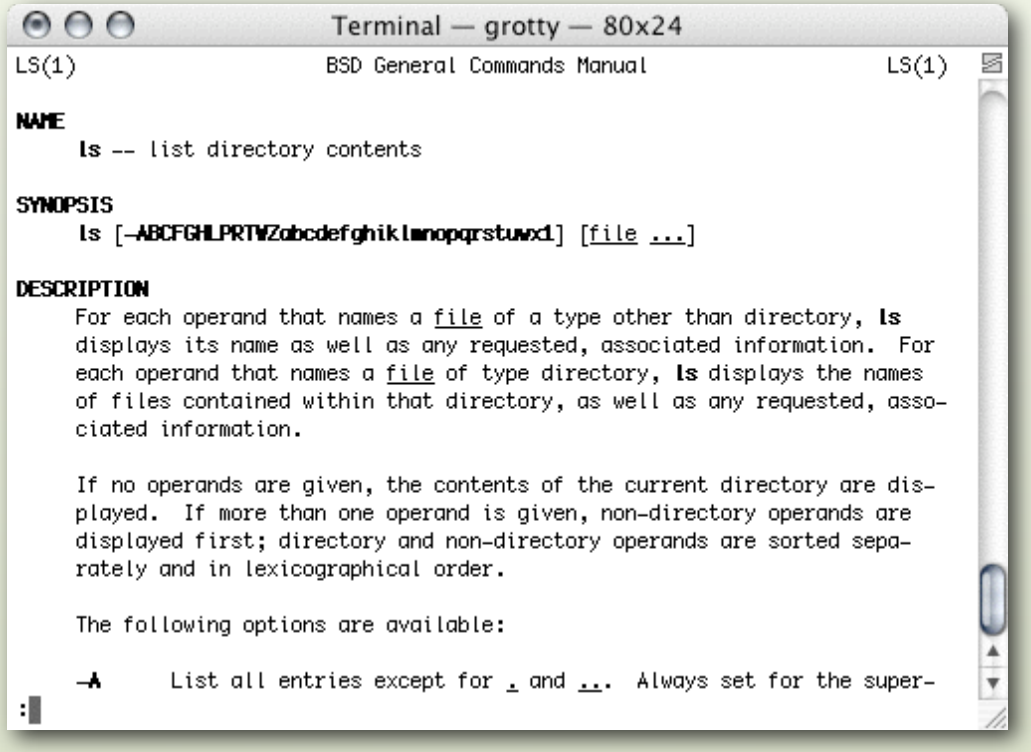
apropos

whatis

locate

find

mdfind



"Don't Memorize Commands, Learn to use the Library"

man page

The Unix manual is organized into **man**ual **page**s

Organized formally to help ensure understanding (MOST of the time...), and usually includes:

NAME	The name of the command
SYNOPSIS	A detailed description of options and arguments • if they are optional • if options must be specified separately, or if they can be combined (and if so, how they can be combined)
DESCRIPTION	Human-readable explanation of what the command does

man page

Organized formally to help ensure understanding (MOST of the time...), and usually includes:

OPTIONS (or COMMAND SUMMARY)	Detailed description of what each of the options do (sometimes folded into the DESCRIPTION)
EXAMPLES	Useful (hopefully!) examples of how to use the command with different options and arguments
DIAGNOSTICS	The return code(s) of the command upon success/failure
ENVIRONMENT	Environment variable usage (we will talk about this...)

man page

Organized formally to help ensure understanding (MOST of the time...), and usually includes:

COMPATIBILITY (or STANDARDS)	How this command meets specific Unix compatibility
SEE ALSO	A list of related commands • "if you like THIS command, you'll LOVE these other commands..."

FILES	Files used by the command for:
	• input
	  • config
	  • data
	• output

man page

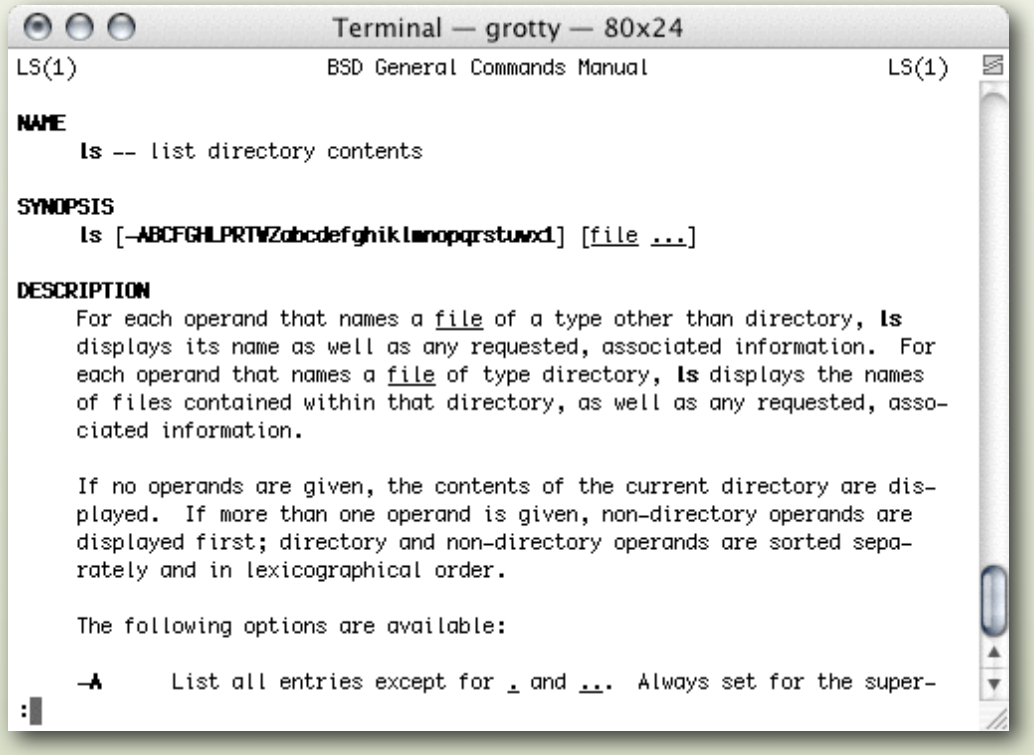
Organized formally to help ensure understanding (MOST of the time...), and usually includes:

HISTORY	Lineage of the command
AUTHORS	Who wrote it • who to blame or who to give thanks!
CAVEATS	Things that may not work as you might expect
BUGS	A synopsis of known bugs in usage, so you don't trip over the command

The `man` page for `ls`

`man` uses a pager
(review from Chapter 3)

- <space bar>
- `b`
- up & down arrows
- `q`
- `/`
- `g` & `G`



`man` page: there's more!

There are nine different **sections** in the `man` pages (section 1 is the default):

- See p. 64 of UfMOSXT
- Can specify section with argument to `man`
  - Example:

```
$ man ls
...
SEE ALSO
     chflags(1), chmod(1), sort(1), xterm(1), termcap(5),
symlink(7), sticky(8)
$ man 8 sticky
...
```

Shell built-in commands are all lumped together on a single `man` page

```
$ man builtin
```

No `man` page...

If no `man` page exists for a command, try executing command with no arguments, or with `-h` or `-help` argument

```
$ man BootCacheControl
No manual entry for BootCacheControl
$ BootCacheControl
BootCacheControl: missing command
Usage: BootCacheControl [-vvv] [-b blocksize] [-f <playlistfile>] start|
stop
            Start/stop the cache using <playlistfile>.
        BootCacheControl statistics
            Print statistics for the currently-active cache.
...
```

- usage: should use the same syntax as a `man` page would
  - if not, curse the programmer...

The Unix "Card Catalog"

What if you don't know what the name of the command is that you want to use?

- chicken-and-egg: can't look at `man` page if you don't know the name of the command!

How do you look things up in the `man` page library?

http://recycledproducts.org/detail.aspx?ID=525

`man -K`

When you give the option `-K` to `man`, it looks for keywords you specify in ALL man pages

Example

```
$ man -K copy
```

...

This is equivalent to not using a "card catalog" at all, but going to the library and starting at the first book, looking in *every* book *in order* until you find what you want

- can be VERY slow

Surely there has to be a better way...

apropos

There is a command that searches through an indexed database of all of the man pages for keywords that you give it called **apropos**

- "Spotlight for the Unix manual" (kind of...)

Example

```
$ apropos copy
...
```

Equivalent (notice it's a lower case `k`, not `K`)

```
$ man -k copy
...
```

whatis

There is another command that searches through the same indexed database as `apropos` called **`whatis`**

- looks for exact match of argument--more restrictive

Example

```
$ whatis copy
...
```

Equivalent

```
$ man -f copy
...
```

locate

It is also possible to search for pathnames (not through `man` pages) to find a match using a command called **`locate`**

- Another aspect of our pseudo-Spotlight capabilities in Unix
- Searches through another indexed database
  - not the same one that `apropos` and `whatis` (and their `man` equivalents) use

Example

```
$ locate copy
...
```

find

The command **find** actually DOES do an active search on the system and does not consult any indexed database

- Even by Unix standards, **find** has arcane syntax--but is VERY powerful
  - **man find**

Example

```
$ find / -name "*jpg"
...
$ find /Applications -perm 0755
...
$ find /Applications -newer myTimedFile.txt
...
$ find / -name “*.app” | less
```

`mdfind` and OS X Spotlight

Searches through the indexed Mac OS X Spotlight **m**eta**d**ata database

- Can search not only by name, but by different criteria
- Database created automatically in the background using `mdimport`
- Database comprised of multiple files:
  - `/.Spotlight-V100`
  - `~/Library/Mail/Envelope Index`
  - `/Volumes/`*`local_volume`*`/.Spotlight-V100`
- Available only on Mac OS X (`locate` and `apropos` are available on other Unices)

For more information, see:

```
$ man mdfind
$ man mdimport
```

Mac OS X CLI tools: ‘‘who’s to thank/blame’’

CLI tools originate from different working groups for Mac OS X

- BSD
- AT&T
- Darwin
- Mac OS
- Shell built-in (specific to shell)
- ?

Questions about generic BSD or shell built-in commands can be directed to BSD or shell information sources

- Mac OS X is a first-class BSD and shell citizen (and as of Leopard, fully UNIX compatible)

Mac OS X CLI tools: ‘‘who’s to thank/blame’’

Darwin or Mac OS specific commands should be directed to Mac OS X-based information sources

- Apple Knowledge Base (formerly known as KBase)
- Apple mailing lists

Commands, Commands, Commands...

Apple Custom Commands

Directory Services

- `dscl`
- `dseditgroup`
- `dsconfigad`
- `dsconfigldap`
- `dscacheutil`
- `dsenableroot`
- `nidump` (obsolete)
- `niload` (obsolete)
- `nicl` (obsolete)
- `lookupd` (obsolete)

Networking

- `ipconfig`
- `atlookup`

Configuration

- `defaults`
- `PlistBuddy`
- `profiles`
- `systemsetup`
- `networksetup`
- `pmset`
- `scutil`

Apple Custom Commands

Filesystem
- `diskutil`
- `hdiutil`
- `SetFile`
- `GetFileInfo`

Spotlight (metadata)
- `mdinfo`
- `mdls`
- `mdfind`
- `mdimport`

Server
- `serversetup`

Security
- `security`
- `certtool`

Deployment/Config
- `asr`

Packages
- `pkgbuild`
- `pkginfo`
- `productbuild`

Launchd
- `launchctl`
- `man launchctl.plist`

Apple Custom Commands

Misc.

- `open`
- `osascript`
- `screencapture`
- `lsbom`
- `installer`
- `softwareupdate`
- `sw_vers`
- `system_profiler`
- `say`

This list is NOT complete...

Third Party Custom Commands

Installed with GUI tools

Purely CLI

- Vendor
- Open Source

Pretty much every computing category you can think of:

- Deployment
- Graphic
- Media
- ...
- (this would be a VERY long list)

Installing Open Source Software

Utilizing Open Source Software on Mac OS X

There is a LOT of open source software out there that has been ported to Mac OS X

- Unless you are already a programmer, you don't want to have to do the porting yourself, because porting can be a royal PITA...

MacPorts & Fink & HomeBrew

The Mac OS X community supports different repositories for open source software that has been ported to Mac OS X:

- MacPorts (formerly known as DarwinPorts)
  - http://www.macports.org/
- Fink
  - http://fink.sourceforge.net/
- HomeBrew
  - http://mxcl.github.com/homebrew/

```
$ ruby -e "$(curl -fsSL https://raw.github.com/mxcl/homebrew/go)"
```

All require Xcode tools (including command line tools) to be installed

Summary

Learning the command line interface is time well spent

It's not a matter of memorizing seemingly arcane, unrelated commands

- there is (mostly) a pattern

It's the gateway drug to Automation...

Resources

MacTech magazine

Apple

- http://manuals.info.apple.com/en_US/IntroCommandLine_v10.6.pdf

Amazon

- Peachpit
- O'Reilly
- ...

acmeFoo CLI Course

- Offered locally (Seattle and Portland), and all over the country/world
- For info email info@acmefoo.org or training@crywolf.com

Google

acmeFoo

A co-op of developers, trainers, & consultants

- "We beat our heads against the wall so you don't have to!"

Goal is to empower you, not merely "can" solutions

Current focuses include:

- Command Line
- Sysadmin Automation
  - Local & Remote (ARD)
- Cocoa (Mac OS X & iOS)
- Mac OS X and iOS Deployment & MDM
- Package Creation
- 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)
  - Coming soon!
- Training sessions offered in Seattle and Portland
  - and across the country (and the world!)

Introduction to the Command-Line Mindset (distilled version...)

acmeFoo

Scott M. Neal
smn.mg@acmefoo.org
MacTech Seattle Mar 2013