

# Introduction to the Command-Line

**MACTECH**

rojoboboto

**echo “\$(whoami)”**

# Why use the Command Line?

Fast  
Powerful  
Interactive  
Scalable

# Troubleshooting Automation Deployment Scripting

More efficient on slow connections  
Benefit from the \*nix Community  
Unique Tools in CLI

You can hang out with the cool kids  
Makes Server Administration a Breeze  
Gateway drug to Linux

But...



Scary for Beginners  
Steep Learning Curve  
Requires Different Way of Thinking

# The Unix Philosophy

This is the Unix philosophy:

Write programs that do one thing and do it well. Write programs to work together. Write programs to handle text streams, because that is a universal interface.

- **Doug McIlroy**,  
(then head of the Bell Labs CSRC)

# The Unix Philosophy

- Small is beautiful.
- Make each program do one thing well.
- Build a prototype as soon as possible.
- Choose portability over efficiency.
- Store data in flat text files.
- Use software leverage to your advantage.
- Use shell scripts to increase leverage and portability.
- Avoid captive user interfaces.
- Make every program a filter.

# The Command-Line Mindset

# Picky

# Rules

# Dig

# Brief History





# Brief History



# A Modern Perspective



# What is the Command Line?



Terminal.app



iTerm.app



## Remote Desktop.app

# 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)

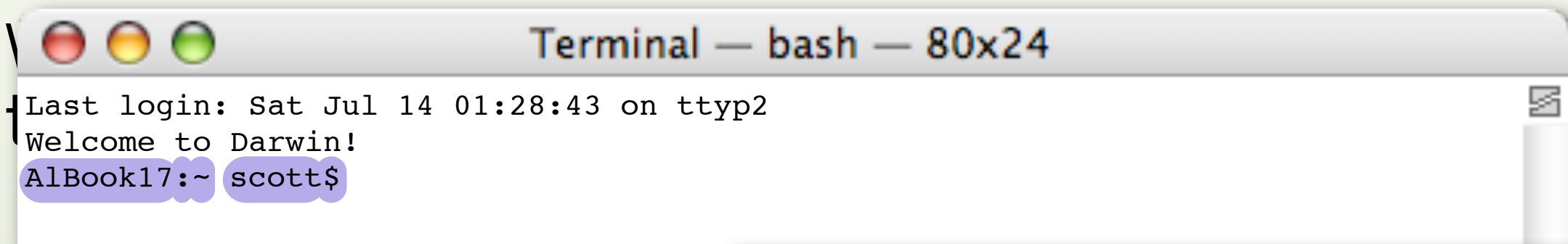


An entry in Directory Services provides default shell (or none) for each user

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



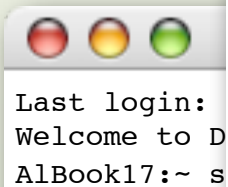
# Interactive Shell Prompt

A screenshot of a macOS Terminal window. The title bar reads "Terminal — bash — 80x24". The window contains the following text: "Last login: Sat Jul 14 01:28:43 on ttty2", "Welcome to Darwin!", and the prompt "AlBook17:~ scott\$". The prompt is highlighted with a light blue background.

```
Terminal — bash — 80x24
Last login: Sat Jul 14 01:28:43 on ttty2
Welcome to Darwin!
AlBook17:~ scott$
```

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 '#'

A smaller screenshot of a Terminal window showing the same login sequence as the larger one above it.

```
Last login:
Welcome to D
AlBook17:~ s
```

# Interactive Shell Prompt

```
rojoair:~ nathan$
```

# Interactive Shell Prompt

\$

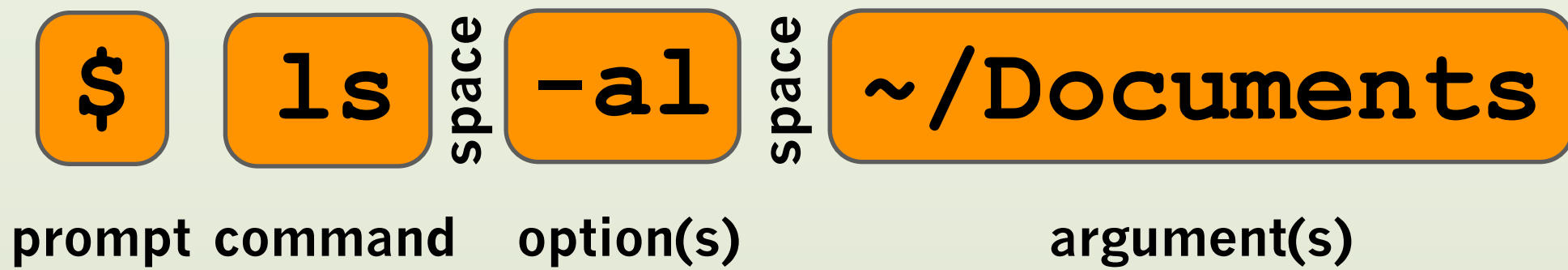
# Unix Command Basics

## some basic commands

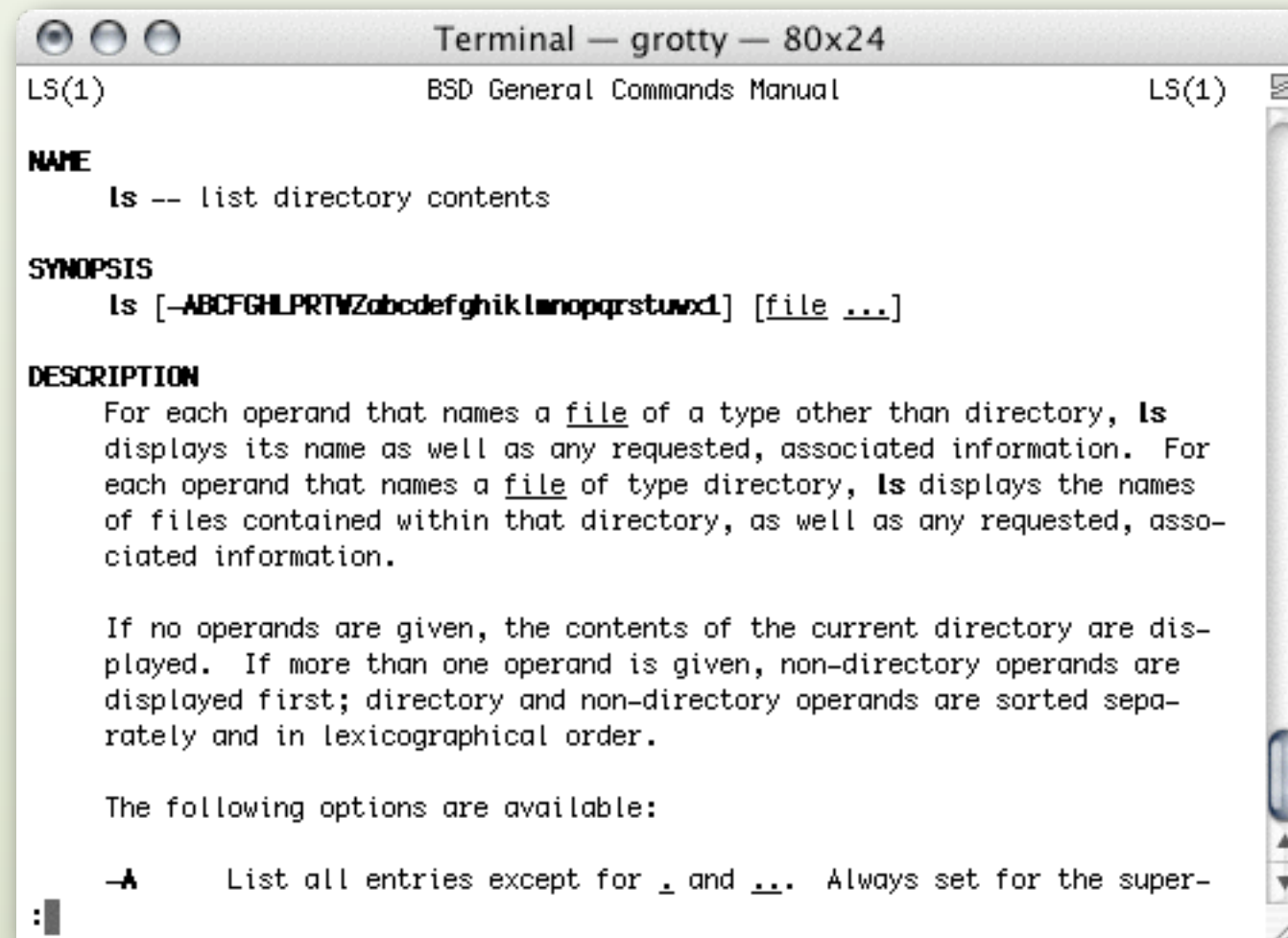
- `ls`
- `cd`
- `cat`
- `grep`
- `echo`

- `mkdir`
- `cp`
- `rm`
- `mv`
- `ssh`

# Anatomy of a command



# Looking things up



A screenshot of a macOS Terminal window titled "Terminal — grotty — 80x24". The window displays the BSD General Commands Manual for the `ls` command. The manual is organized into sections: NAME, SYNOPSIS, and DESCRIPTION. The NAME section states that `ls` is used to list directory contents. The SYNOPSIS section shows the command syntax: `ls [-ABCFGHLPRTVZabcdefghiklnopqrstuvwX] [file ...]`. The DESCRIPTION section explains that `ls` displays file names and associated information, and that if no operands are given, it lists the current directory. It also mentions that operands are sorted separately and in lexicographical order. At the bottom, it lists available options, starting with `-A` for listing all entries except for `.` and `..`.

```
Terminal — grotty — 80x24
LS(1) BSD General Commands Manual LS(1)

NAME
  ls -- list directory contents

SYNOPSIS
  ls [-ABCFGHLPRTVZabcdefghiklnopqrstuvwX] [file ...]

DESCRIPTION
  For each operand that names a file of a type other than directory, ls
  displays its name as well as any requested, associated information. For
  each operand that names a file of type directory, ls displays the names
  of files contained within that directory, as well as any requested, asso-
  ciated information.

  If no operands are given, the contents of the current directory are dis-
  played. If more than one operand is given, non-directory operands are
  displayed first; directory and non-directory operands are sorted sepa-
  rately and in lexicographical order.

  The following options are available:

  -A      List all entries except for . and ... Always set for the super-
```

“Don’t Memorize Commands, Learn to use the Library”



man

let's dive in!

# output redirect



let's dive in!

# 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

# 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?

let's dive in!



# 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 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

# Command Line Grammar and Syntax

su & sudo

# 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 **substitute user** (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 **d**o)

- 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

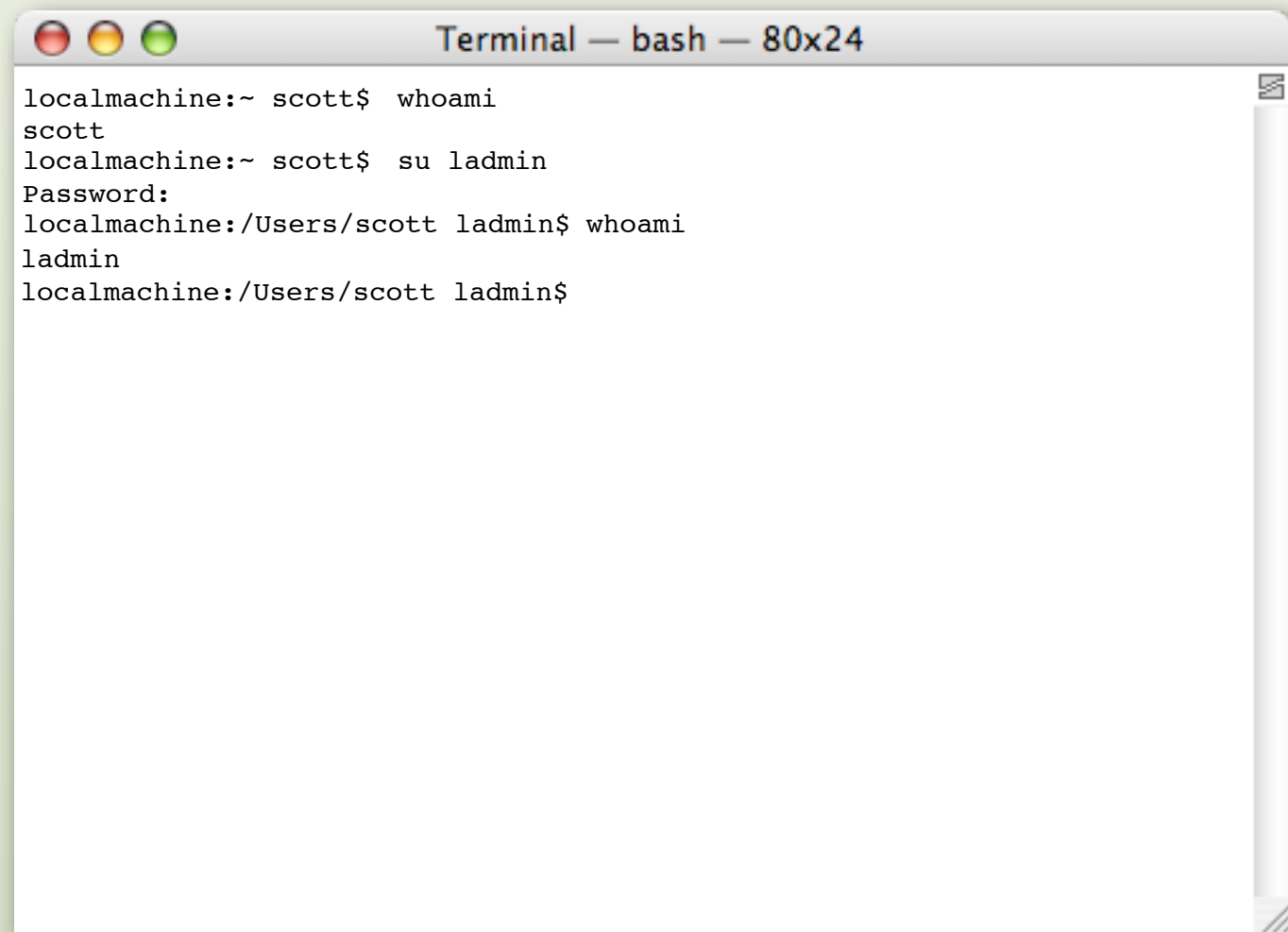
- but you can `su` to an admin user first, and *then* `sudo`

# Who Am I?

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

There is a command to help you

\$ `whoami`

A screenshot of a macOS Terminal window titled "Terminal — bash — 80x24". The window shows a sequence of commands and their outputs. First, the user runs "whoami" as "scott", which outputs "scott". Then, the user runs "su ladmin", which prompts for a password. After the password is entered, the prompt changes to "localmachine:/Users/scott ladmin\$". Finally, the user runs "whoami" again, which outputs "ladmin".

```
localmachine:~ scott$ whoami
scott
localmachine:~ scott$ su ladmin
Password:
localmachine:/Users/scott ladmin$ whoami
ladmin
localmachine:/Users/scott ladmin$
```

# 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!



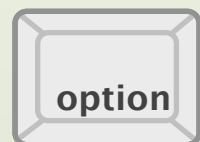
# Modifier Keys

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



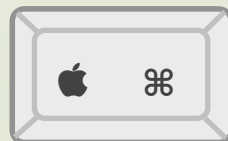
^

Control



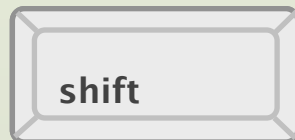
⌥

Option (sometimes alt)



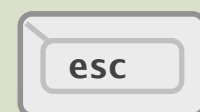
⌘

Command (sometimes Apple)



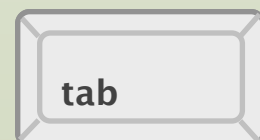
⇧

Some important CLI keys are not modifier keys



⌫

Escape



→|

Tab (forward)

# Shell Shortcuts

Tab completion	<p>Shell will automatically complete the file path for you</p> <ul style="list-style-type: none"><li>• If you haven't given it a typo!</li><li>• If there is more than one choice, use double-Tab to show the multiple choices</li><li>• Case-sensitive (even though file system itself usually isn't)</li></ul>
Left & Right Arrow keys	Move cursor without deleting any characters
Up & Down Arrow keys	Show previous lines entered into the shell

# Shell Shortcuts

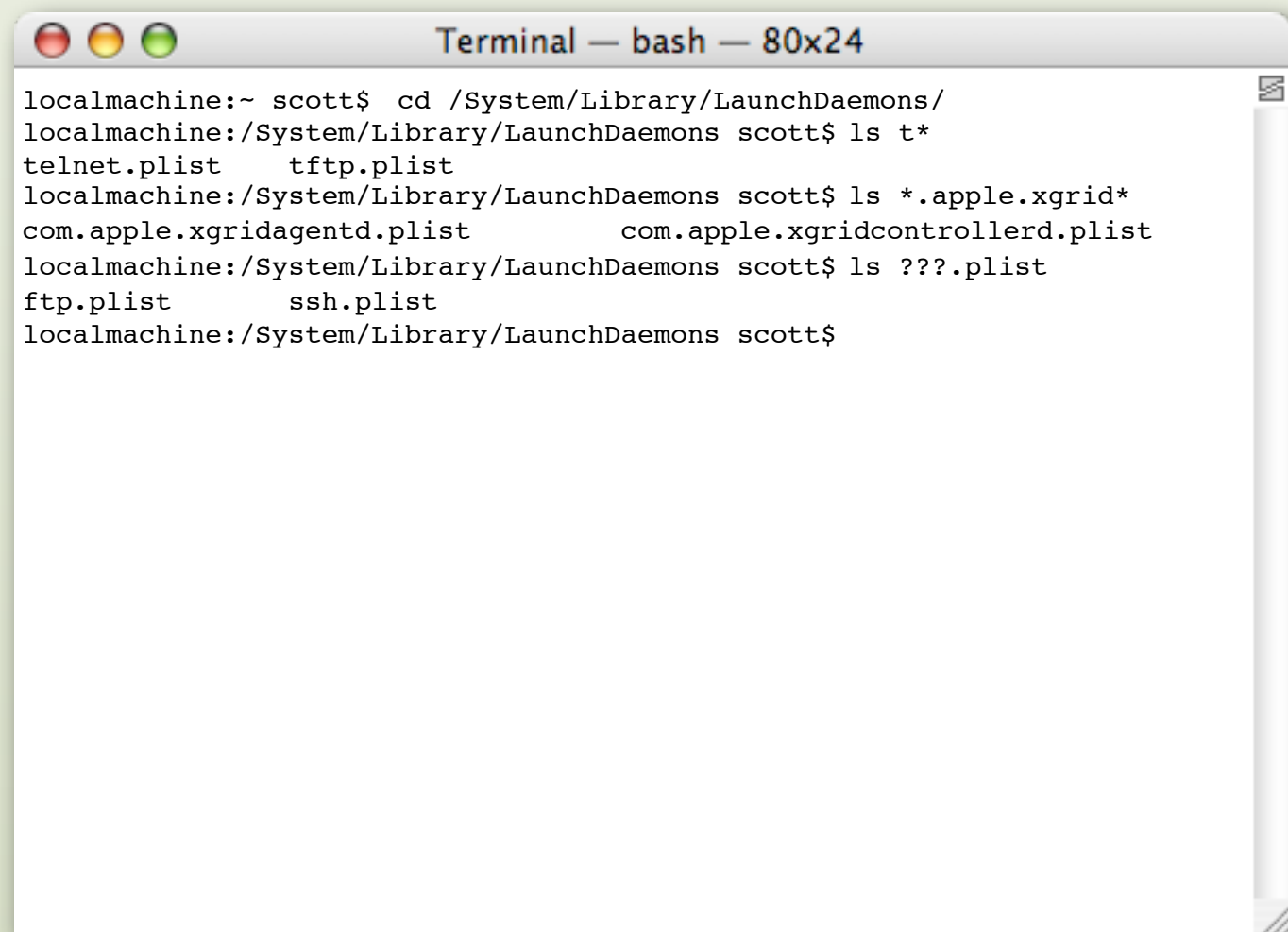
Delete key or ^H	Delete character to left of cursor, move remaining characters to the left
^A	Move to beginning of line
^E	Move to <b>E</b> nd of line
^L	<b>c</b> lear entire Shell screen by blanking out lines
^U	<b>U</b> ntype from cursor to beginning of line
^K	Clear from cursor to end of line

# Wildcard characters

The asterisk ‘\*’ is a wildcard that matches anything.

The question mark ‘?’ matches any single character.

Wildcards characters are sometimes called **globbing** characters

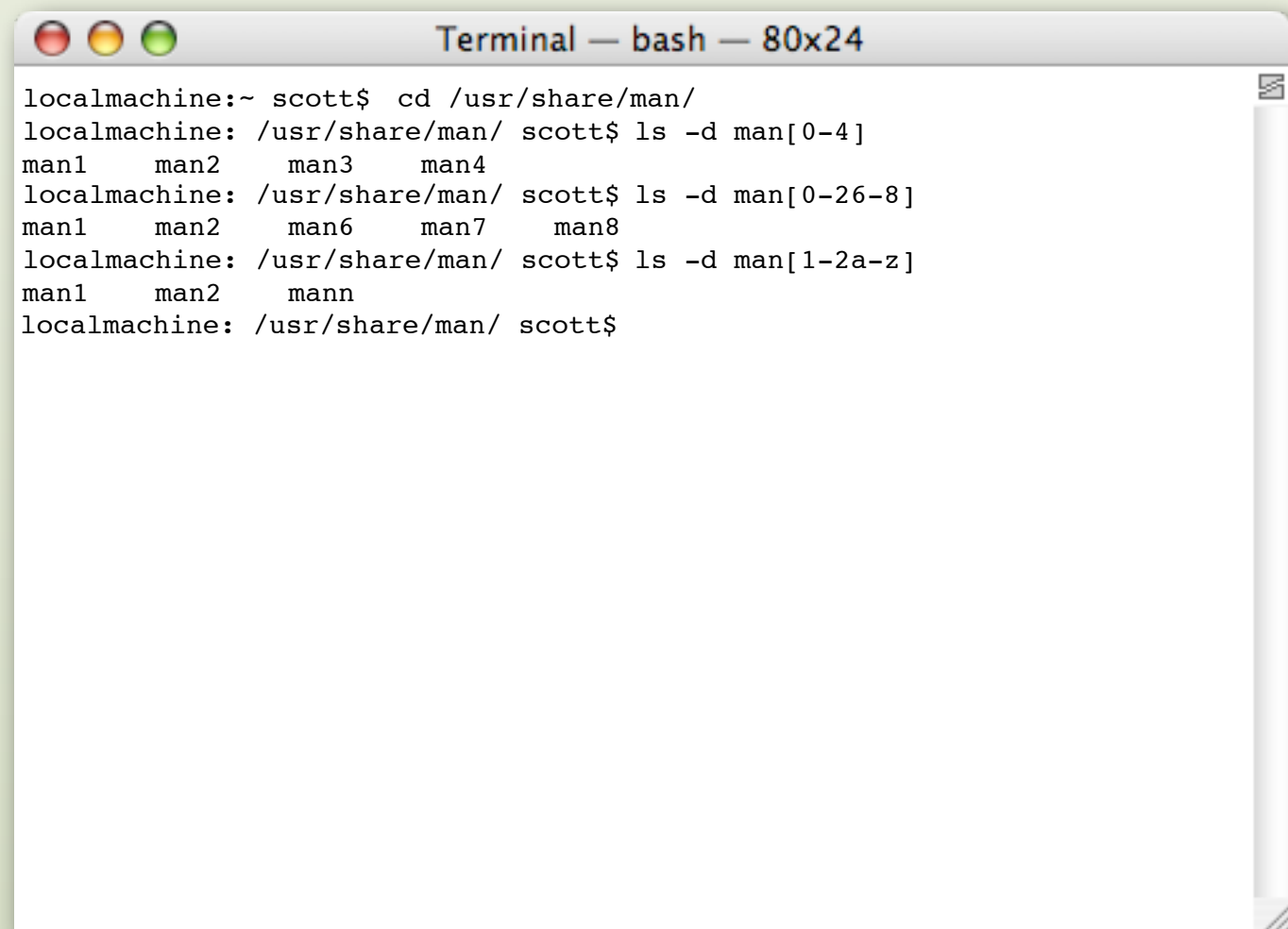
A screenshot of a macOS Terminal window titled "Terminal — bash — 80x24". The window shows a series of commands and their outputs in a monospaced font. The user navigates to the directory /System/Library/LaunchDaemons/ and uses the 'ls' command with various wildcards to list files. The output shows files like telnet.plist, tftp.plist, and several .apple.xgrid\* files, as well as files ending in .plist.

```
localmachine:~ scott$ cd /System/Library/LaunchDaemons/  
localmachine:/System/Library/LaunchDaemons scott$ ls t*  
telnet.plist      tftp.plist  
localmachine:/System/Library/LaunchDaemons scott$ ls *.apple.xgrid*  
com.apple.xgridagentd.plist      com.apple.xgridcontrollerd.plist  
localmachine:/System/Library/LaunchDaemons scott$ ls ????.plist  
ftp.plist      ssh.plist  
localmachine:/System/Library/LaunchDaemons scott$
```

# 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

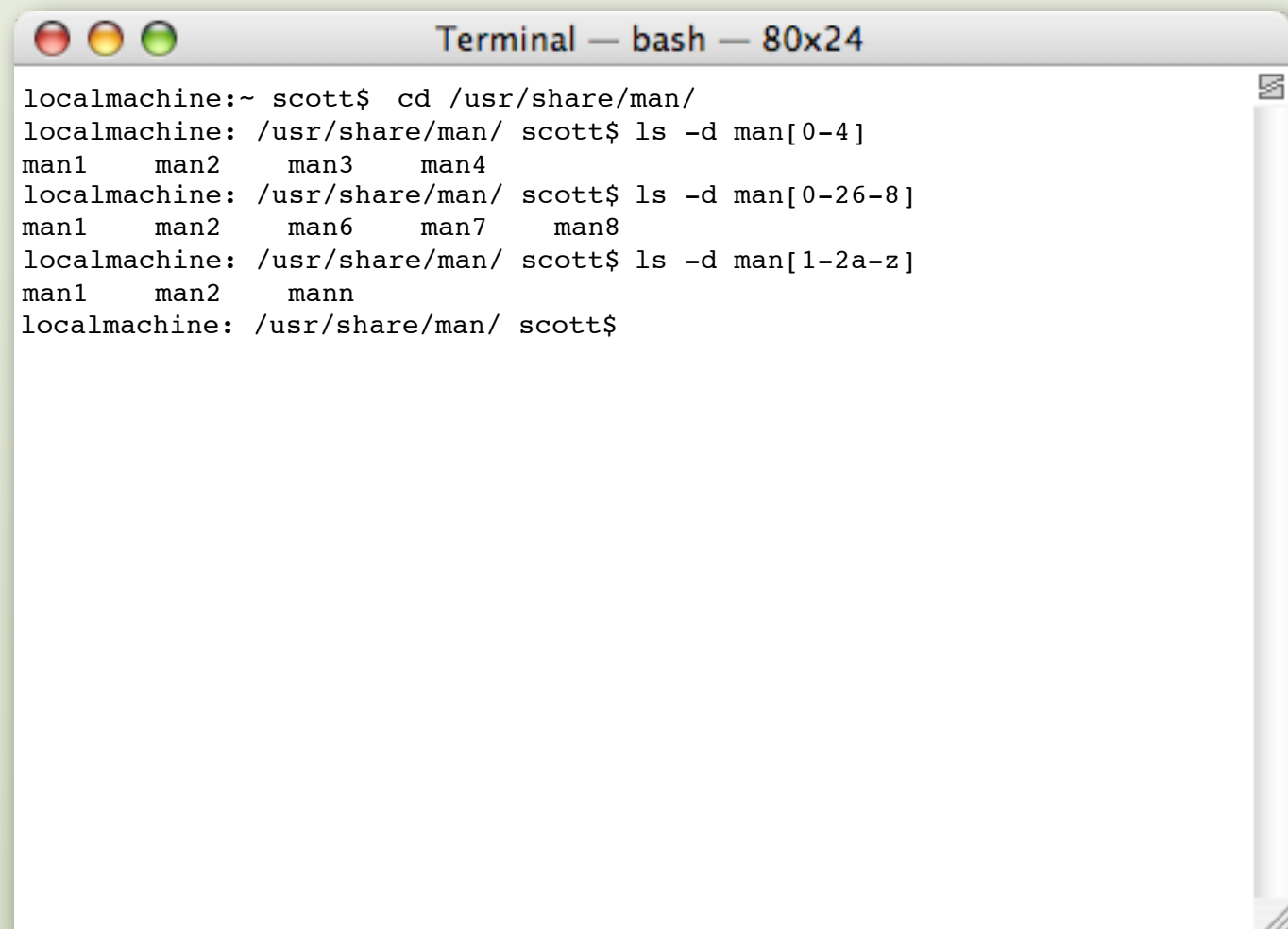
A terminal window titled "Terminal — bash — 80x24" showing a series of commands and their outputs. The user navigates to the directory /usr/share/man/ and uses the 'ls' command with various wildcard patterns to list files. The first command lists files man[0-4], showing man1 through man4. The second command lists files man[0-26-8], showing man1 through man8. The third command lists files man[1-2a-z], showing man1, man2, and mann.

```
localmachine:~ scott$ cd /usr/share/man/
localmachine: /usr/share/man/ scott$ ls -d man[0-4]
man1    man2    man3    man4
localmachine: /usr/share/man/ scott$ ls -d man[0-26-8]
man1    man2    man6    man7    man8
localmachine: /usr/share/man/ scott$ ls -d man[1-2a-z]
man1    man2    mann
localmachine: /usr/share/man/ scott$
```

# 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

A terminal window titled "Terminal — bash — 80x24" showing a series of commands and their outputs. The user is in the directory /usr/share/man/. The commands and outputs are: 1. Command: cd /usr/share/man/; Output: localmachine:~ scott\$. 2. Command: ls -d man[0-4]; Output: localmachine: /usr/share/man/ scott\$; man1 man2 man3 man4. 3. Command: ls -d man[0-26-8]; Output: localmachine: /usr/share/man/ scott\$; man1 man2 man6 man7 man8. 4. Command: ls -d man[1-2a-z]; Output: localmachine: /usr/share/man/ scott\$; man1 man2 mann. 5. Command: (no output shown). 6. Command: (no output shown). 7. Command: (no output shown).

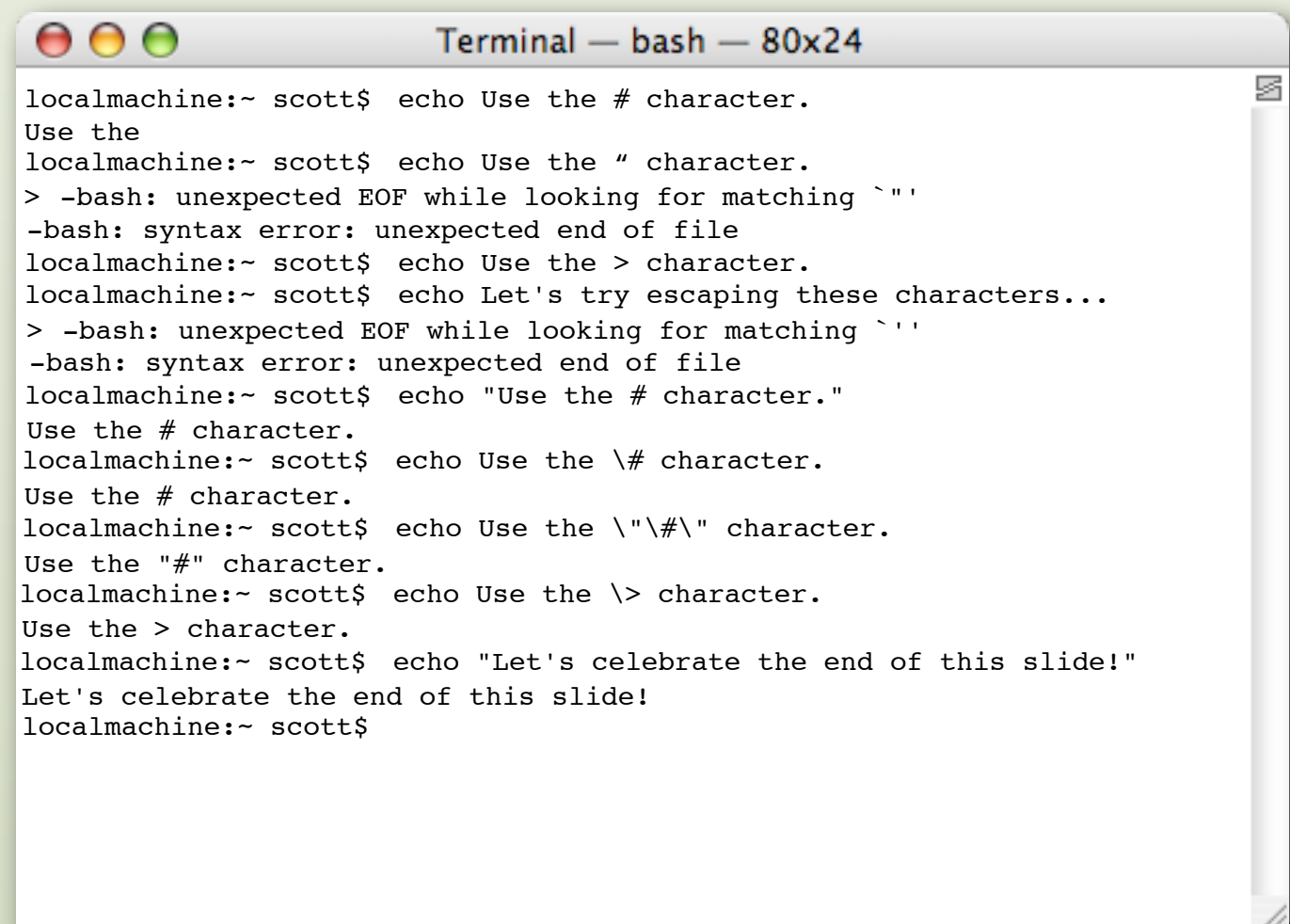
```
localmachine:~ scott$ cd /usr/share/man/
localmachine: /usr/share/man/ scott$ ls -d man[0-4]
man1    man2    man3    man4
localmachine: /usr/share/man/ scott$ ls -d man[0-26-8]
man1    man2    man6    man7    man8
localmachine: /usr/share/man/ scott$ ls -d man[1-2a-z]
man1    man2    mann
localmachine: /usr/share/man/ scott$
```

# 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.

A terminal window titled "Terminal — bash — 80x24" showing a series of commands and their outputs. The user 'scott' is at a 'localmachine' prompt. The commands demonstrate the use of special characters like #, ", >, and \, and how they are escaped. The final command is a celebratory message.

```
localmachine:~ scott$ echo Use the # character.  
Use the  
localmachine:~ scott$ echo Use the " character.  
> -bash: unexpected EOF while looking for matching `"  
-bash: syntax error: unexpected end of file  
localmachine:~ scott$ echo Use the > character.  
localmachine:~ scott$ echo Let's try escaping these characters...  
> -bash: unexpected EOF while looking for matching `'  
-bash: syntax error: unexpected end of file  
localmachine:~ scott$ echo "Use the # character."  
Use the # character.  
localmachine:~ scott$ echo Use the \# character.  
Use the # character.  
localmachine:~ scott$ echo Use the \"#\#\" character.  
Use the "#\" character.  
localmachine:~ scott$ echo Use the \> character.  
Use the > character.  
localmachine:~ scott$ echo "Let's celebrate the end of this slide!"  
Let's celebrate the end of this slide!  
localmachine:~ scott$
```

# 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

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**

# man page

The Unix manual is organized into **manual pages**

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

NAME	The name of the command
SYNOPSIS	<p>A detailed description of options and arguments</p> <ul style="list-style-type: none"><li>• if they are optional</li><li>• if options must be specified separately, or if they can be combined (and if so, how they can be combined)</li></ul>
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 <ul style="list-style-type: none"><li>• “if you like THIS command, you’ll LOVE these other commands...”</li></ul>
FILES	Files used by the command for: <ul style="list-style-type: none"><li>• input<ul style="list-style-type: none"><li>• config</li><li>• data</li></ul></li><li>• output</li></ul>



# man page

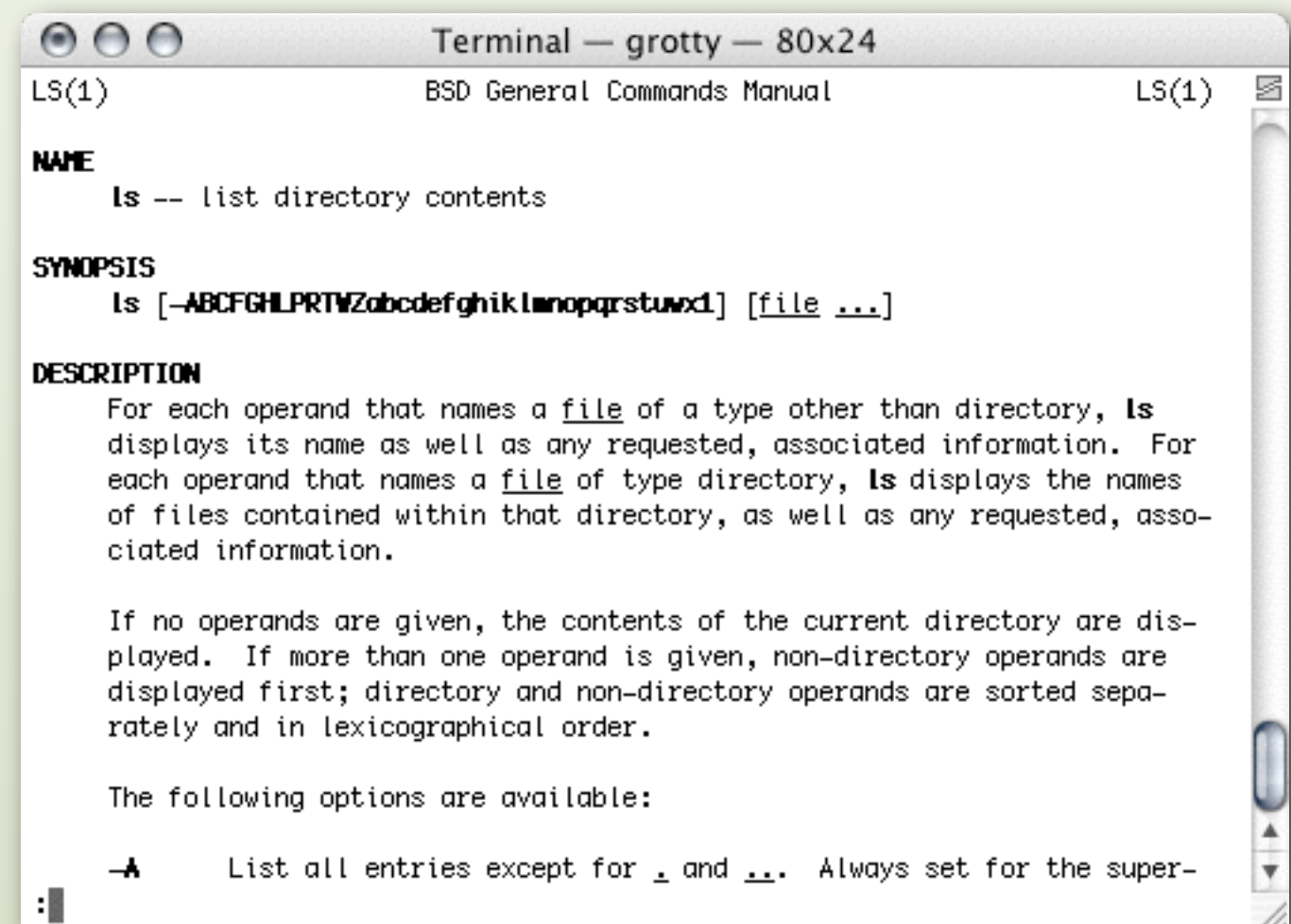
Organized formally to help ensure understanding (MOST of the time...), and usually includes:

HISTORY	Lineage of the command
AUTHORS	Who wrote it <ul style="list-style-type: none"><li>• who to blame or who to give thanks!</li></ul>
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



```
Terminal — grotty — 80x24
LS(1) BSD General Commands Manual LS(1)

NAME
  ls -- list directory contents

SYNOPSIS
  ls [-ABCFGHLPRTVZabcdefghiklnopqrstuwxd] [file ...]

DESCRIPTION
  For each operand that names a file of a type other than directory, ls
  displays its name as well as any requested, associated information. For
  each operand that names a file of type directory, ls displays the names
  of files contained within that directory, as well as any requested, asso-
  ciated information.

  If no operands are given, the contents of the current directory are dis-
  played. If more than one operand is given, non-directory operands are
  displayed first; directory and non-directory operands are sorted sepa-
  rately and in lexicographical order.

  The following options are available:

  -A      List all entries except for . and ... Always set for the super-
```

# 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...

# what is

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

- looks for exact match of argument--more restrictive

Example

```
$ what is copy
```

...

Equivalent

```
$ man -f 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 **metadata** 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
$ man mdls
$ mdfind "kMDItemKind" = 'Application'
```

# 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

# 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
- screenshot
- lsbon
- installer
- softwareupdate
- sw\_vers
- system\_profiler
- say

This list is NOT complete...

# Third Party Custom Commands

# Package Managers



# MacPorts\* & HomeBrew\*

- **MacPorts** (formerly known as DarwinPorts)
  - <http://www.macports.org/>
- **HomeBrew** (my favorite)
  - <http://mxcl.github.com/homebrew/>

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

\*Requires Xcode tools (including command line tools) to be installed