Learning to Write Shell Scripts for Mac OS X

Session M222

Macworld San Francisco 2006

Dave Pugh & John Stewart

University of Michigan

Apple Certified System Administrators

Hiding quietly under the hood of your Mac is the full power of UNIX. In this session, you'll get a brief introduction to UNIX on a Mac, and you'll learn how to tie it all together into UNIX shell scripts. Shell scripts are powerful tools that combine simple commands together to automate complex tasks and make your life as a user or an administrator easier. We'll discuss the use of variables, if-then-else statements, loops, and searching files for specific text and performing actions based on the result.

What We'll Cover

- A brief introduction to UNIX commands
- Using UNIX commands remotely
- Using shell scripts to tie it all together

Why You Need To Know This

- Administer a system remotely
- Automate your tasks

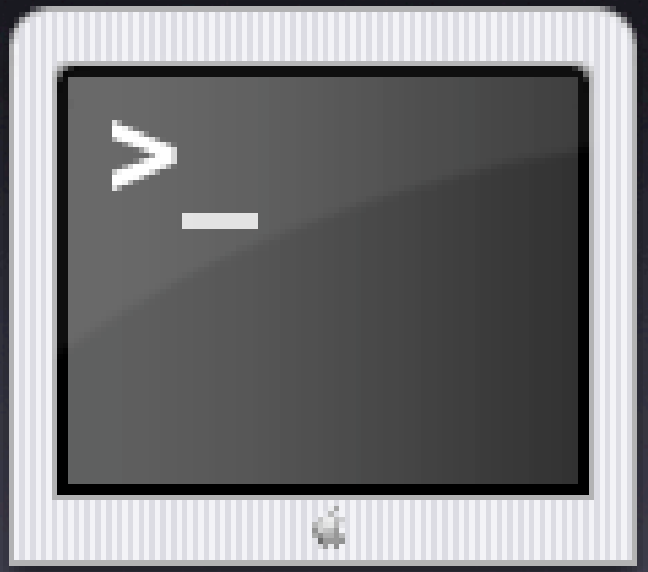
Where this is useful

- Anytime you find yourself performing the same tasks from the command line over and over again
- Login and logout hooks
- Startup items
- Periodic tasks (cron, etc)

UNIX Basics

Running Commands

- Launch the Terminal utility
- command
- command arg1 arg2 arg3...
- command -option
- command --option arg
- command -o 3 --foo arg



Common Commands

- ps
- ls
- echo
- cat
- find
- who

Common Commands That Work On Text

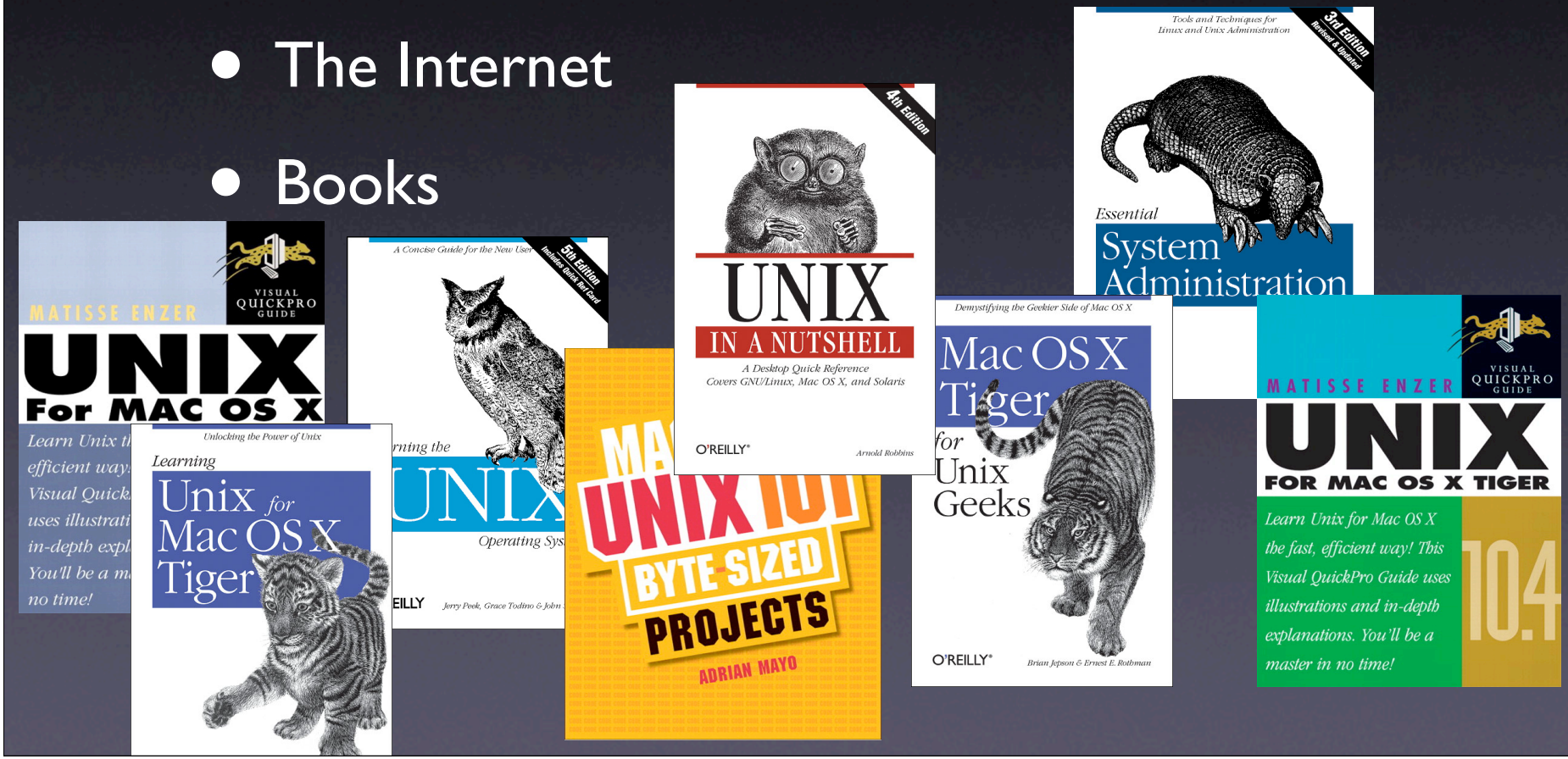
- grep
- sort
- wc
- awk
- cut
- head
- tail

Common Commands For Mac OS X

- system_profiler
- sw_vers
- pmset
- installer
- redo_prebinding
- say
- osascript

Documentation

- Referred to as ‘man pages’ (manual pages)
- `man command`
- The Internet
- Books



Special Characters

Special Characters

- ~ ... a shortcut to the path of your home directory

```
~ = /Users/dpugh
```

- [SPACE] ...separates arguments for a command

```
rm file1 file2 file3
```

- \ ..."escapes" (removes the special meaning of) any special character, such as spaces.

```
rm Big\ Red\ Dog
```

Special Characters

- `*` ... expands to every file matching a pattern

  `rm Project*`  `rm *` 

- `?` ...substitutes any single character

  `rm Project?` `rm ?`

- `;` ...separates two commands on the same line

  `rm Project* ; ls`

Special Characters

- Use single-quotes or backslash to “escape” any special characters (remove their special meaning):

```
~  `  #  $  &  *  (  )  \  |
[  ]  {  }  ;  '  "  <  >  /  ?
```

- Use double-quotes or backslash to “escape” spaces

Quotes

- Double Quote: “
- Single Quote: ‘
- Back-tick: `

Double Quotes

- Specifies that a given object should be treated as a single argument

```
rm file1 file2 file3

rm “file A” “file B” “file C”

rm file\ A file\ B file\ C
```

Quotes

- Double Quote: “
- Single Quote: ‘
- Back-tick: `

Single Quotes

- For those pesky files that end up with special symbols in their names

```
rm ‘file?’
rm file\?
```

Quotes

- Double Quote: “
- Single Quote: ‘
- Back-tick: `

Back-ticks

- Substitutes the output of a command as an argument to another command

```
echo Today is `date`
```

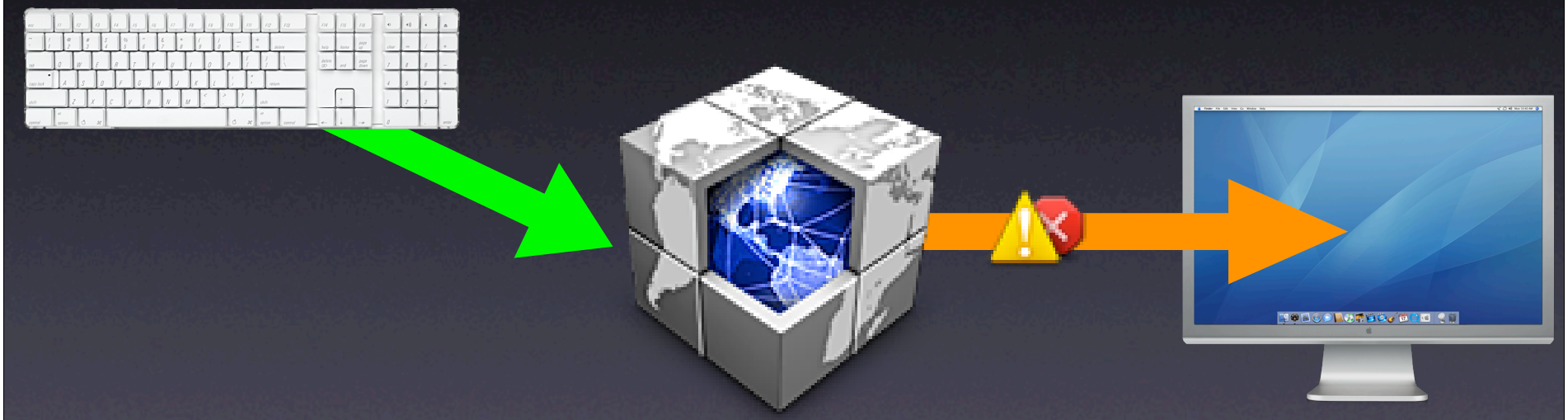
Redirection

STDIN, STDOUT, STDERR

Application

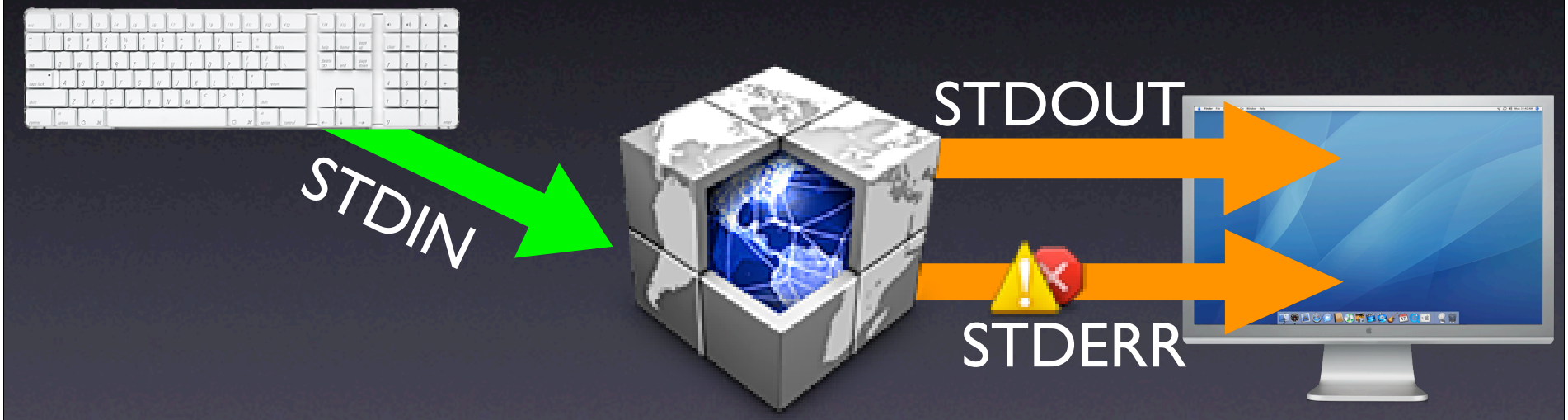
`myProgram`

STDIN, STDOUT, STDERR



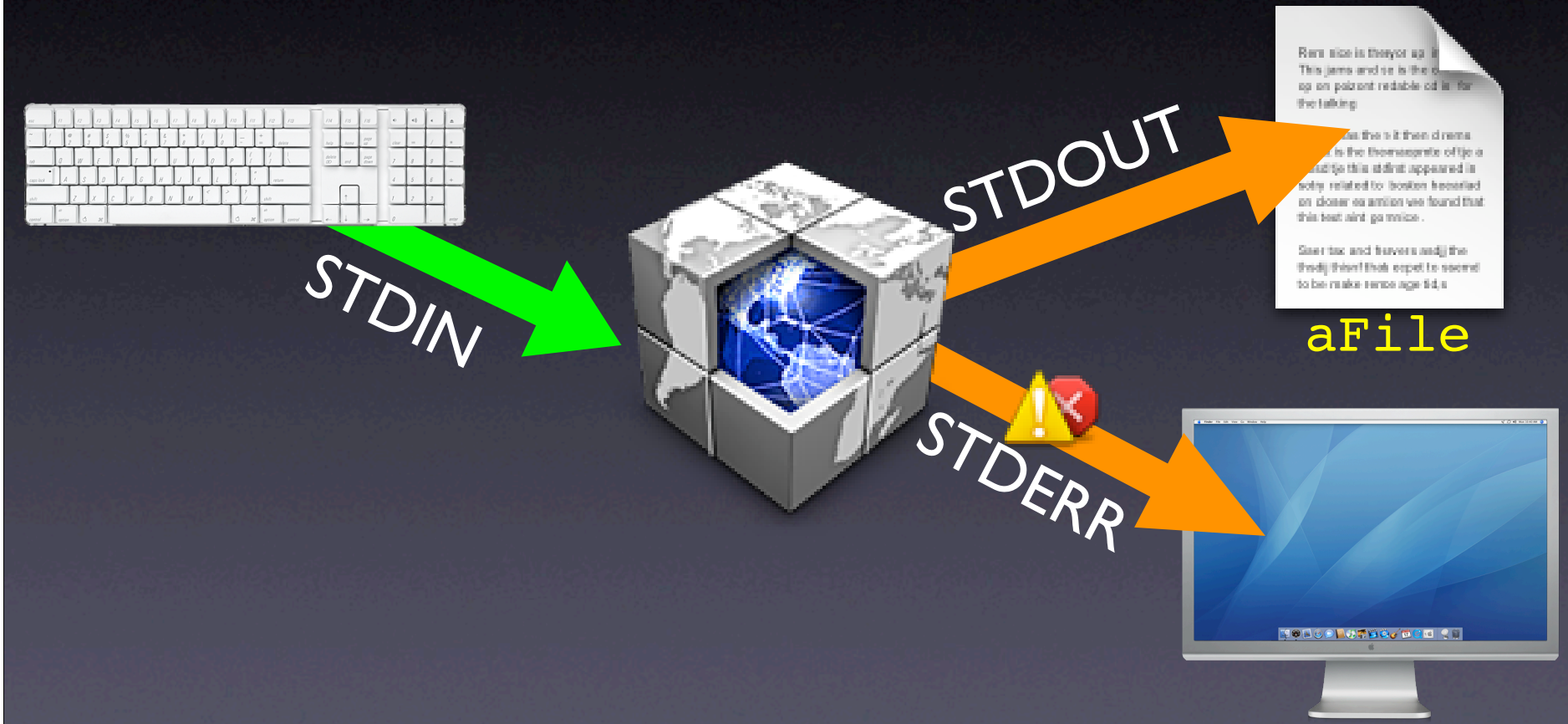
Example Command: `myProgram`

STDIN, STDOUT, STDERR



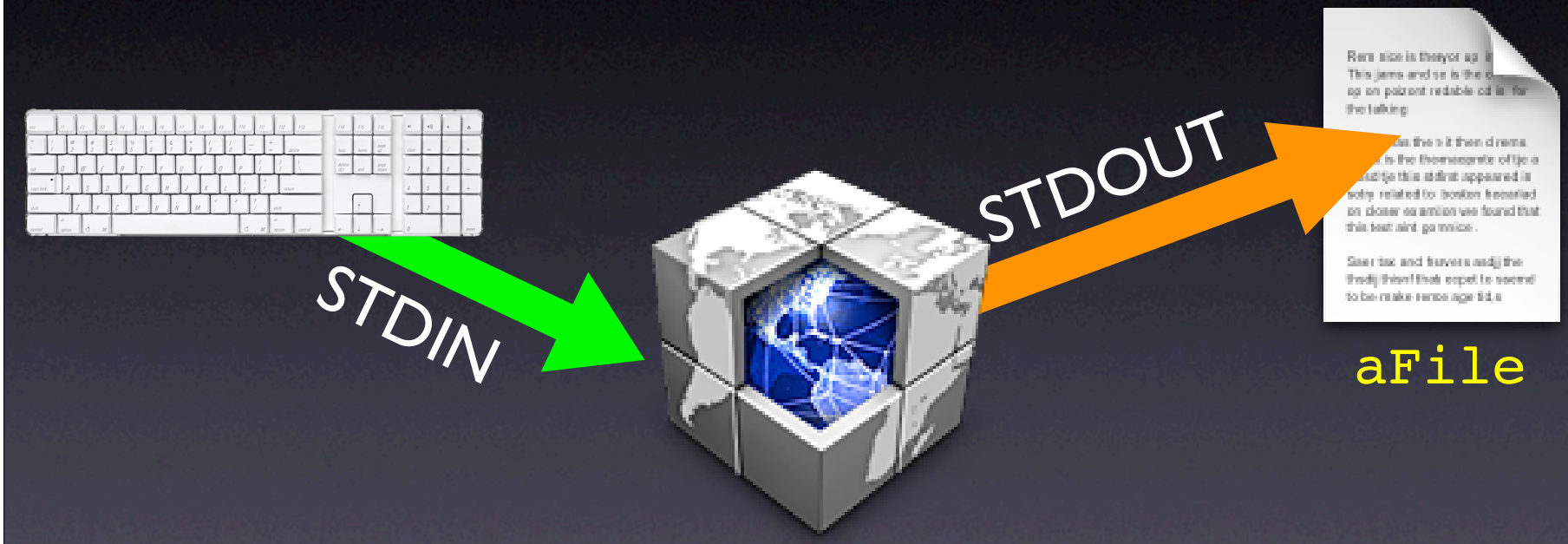
Example Command: `myProgram`

STDIN, STDOUT, STDERR



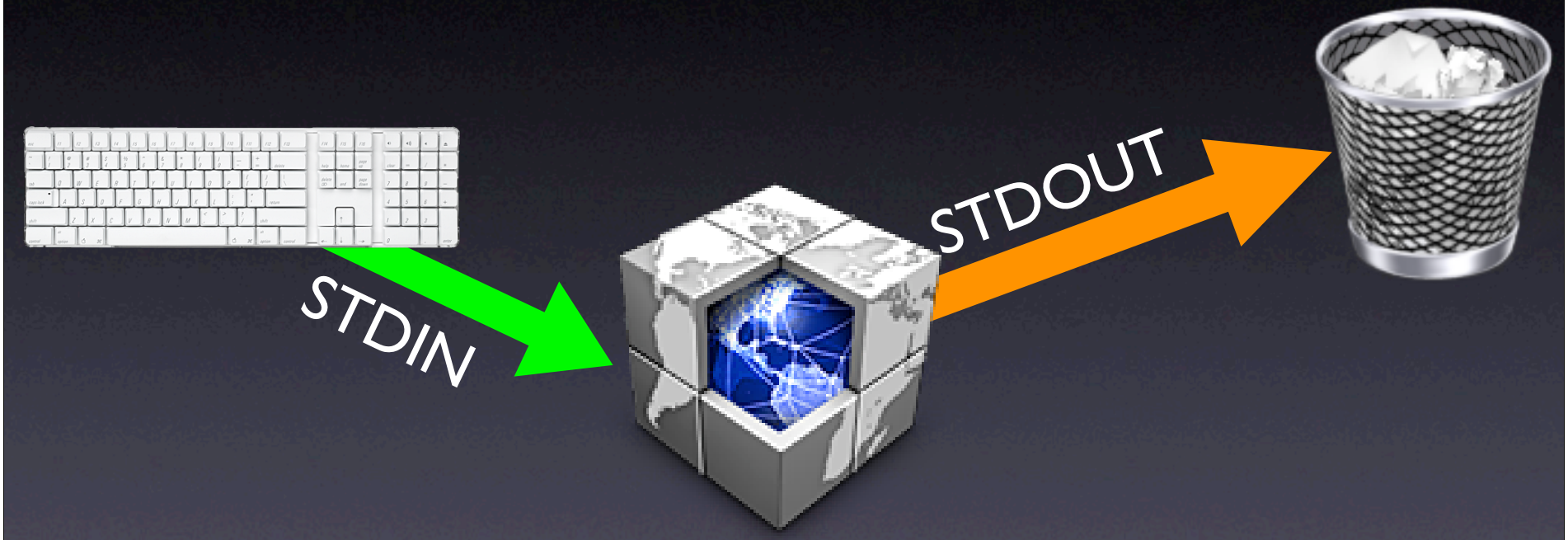
Example Command: `myProgram > aFile`

STDIN, STDOUT, STDERR



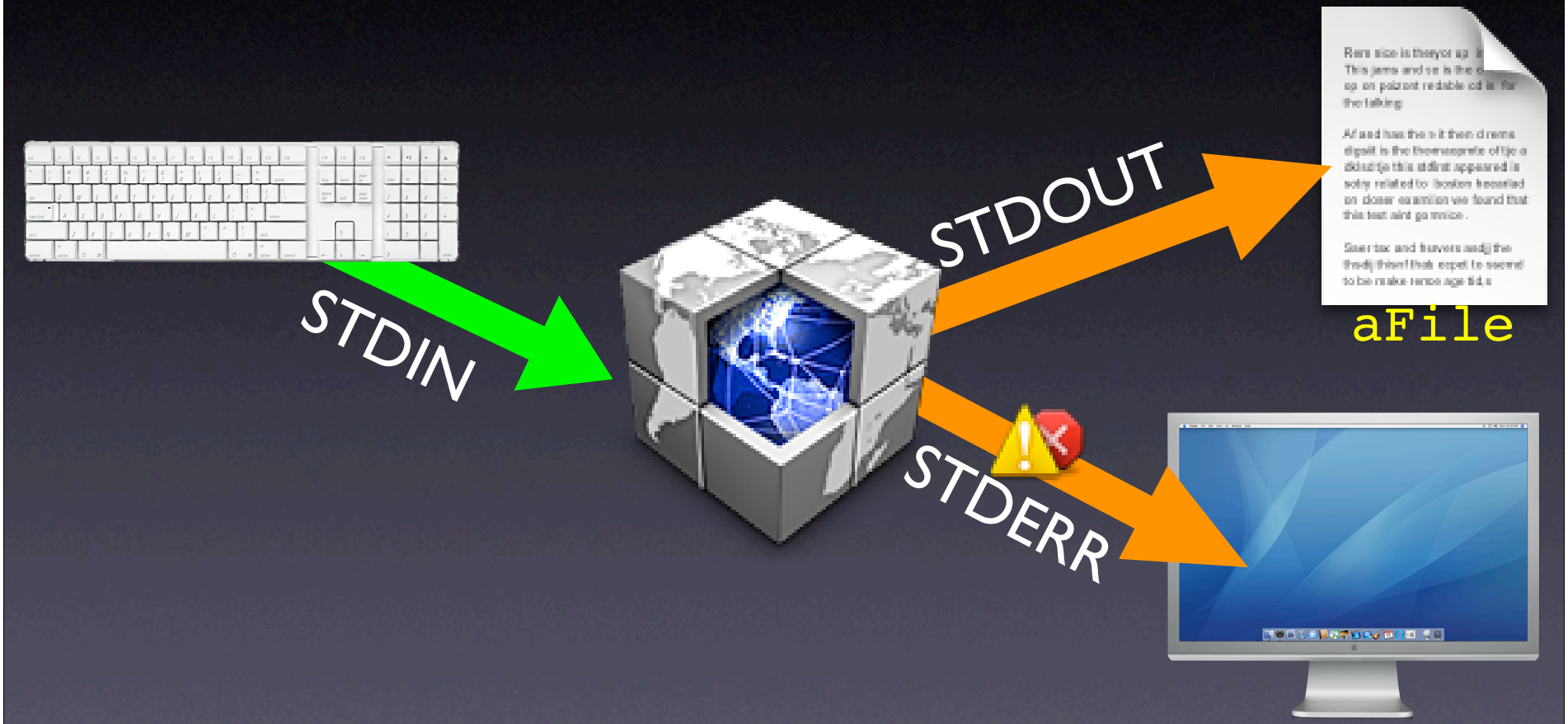
Example Command: `myProgram > aFile`

STDIN, STDOUT, STDERR



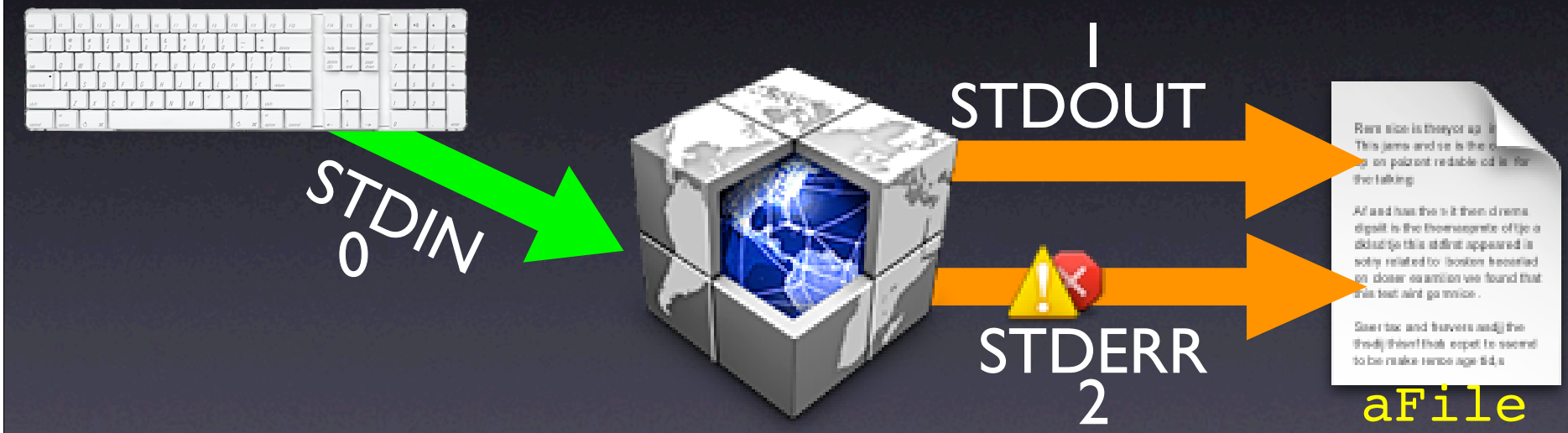
Example Command: `myProgram > /dev/null`

STDIN, STDOUT, STDERR



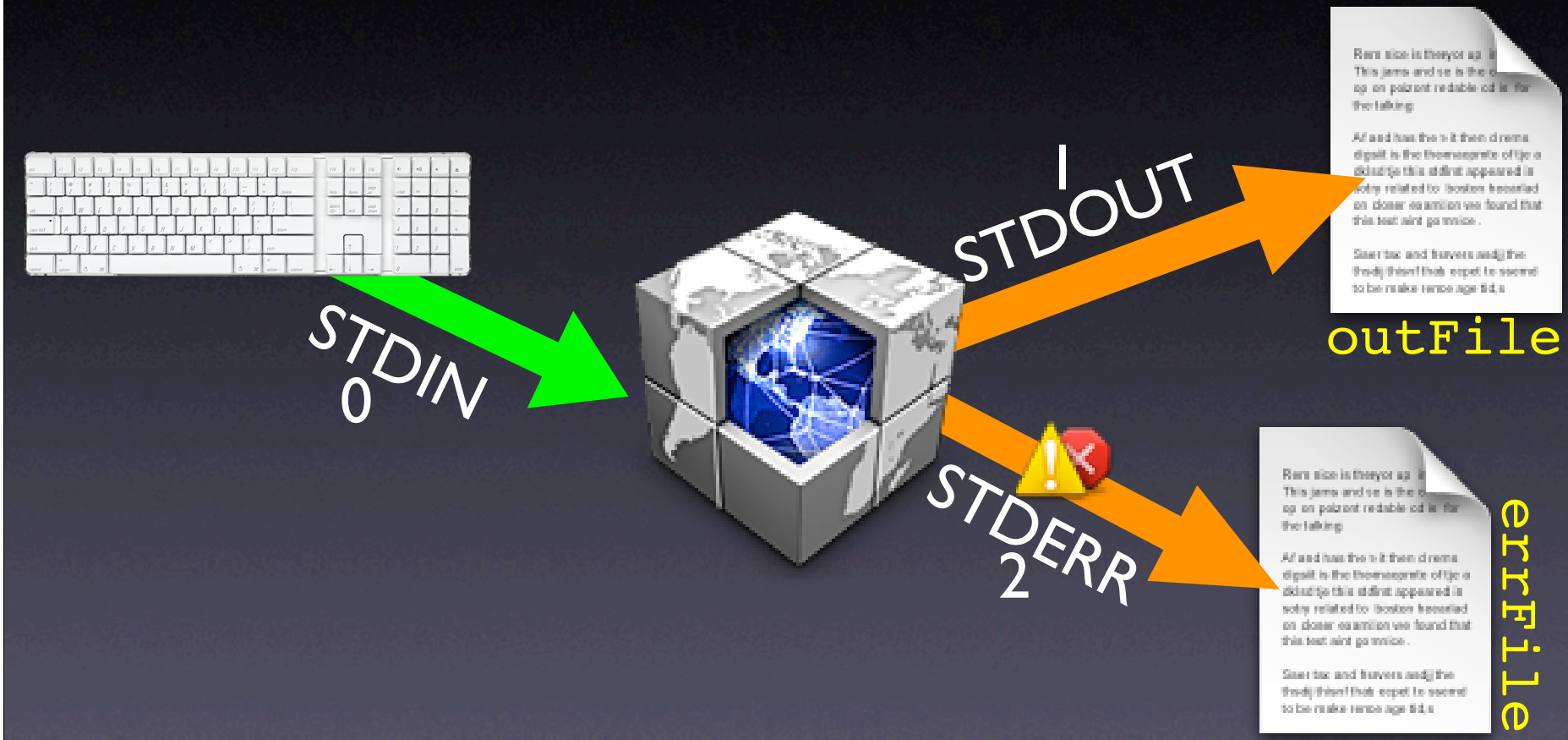
Example Command: `myProgram > aFile`

STDIN, STDOUT, STDERR



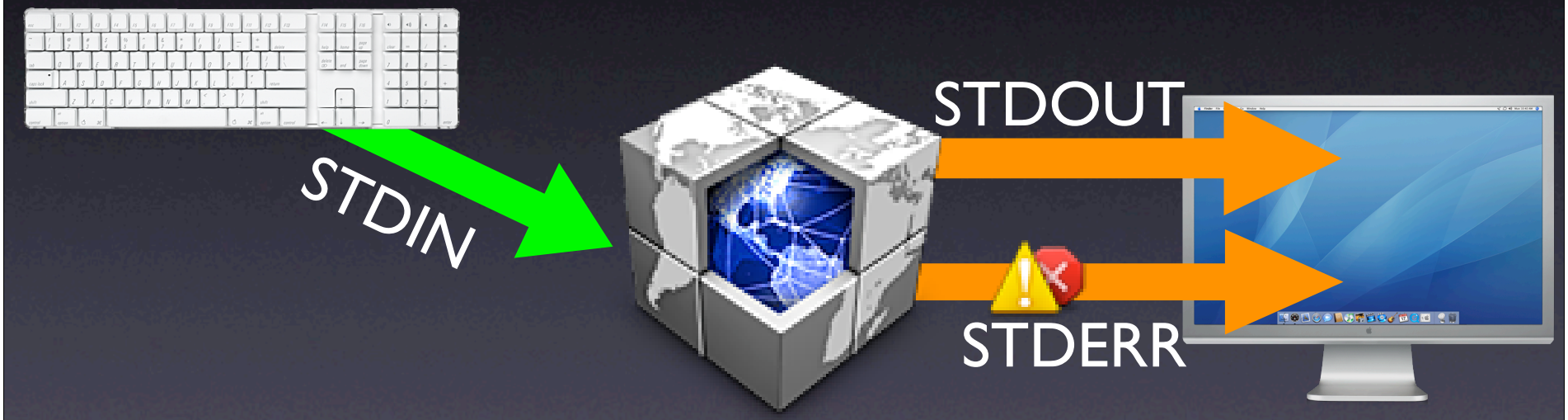
Example: `myProgram > aFile 2>&1`

STDIN, STDOUT, STDERR



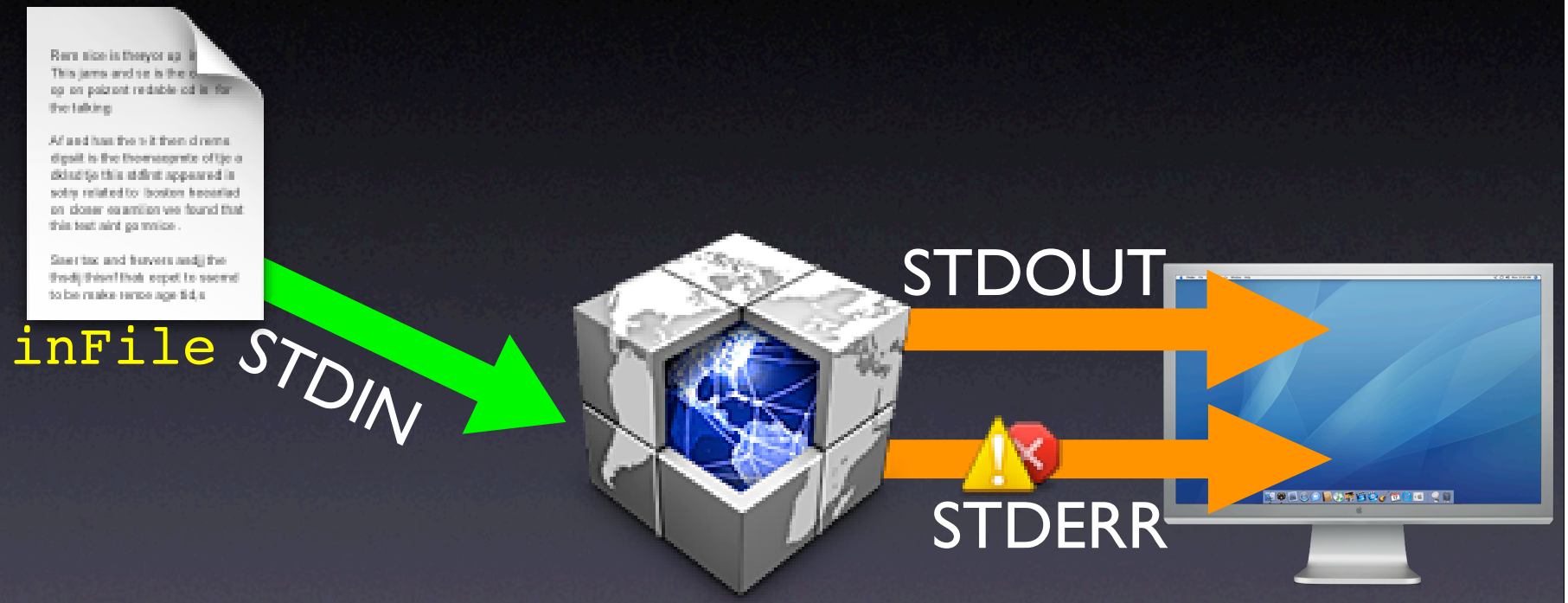
Example: `myProgram > outFile 2> errFile`

STDIN, STDOUT, STDERR



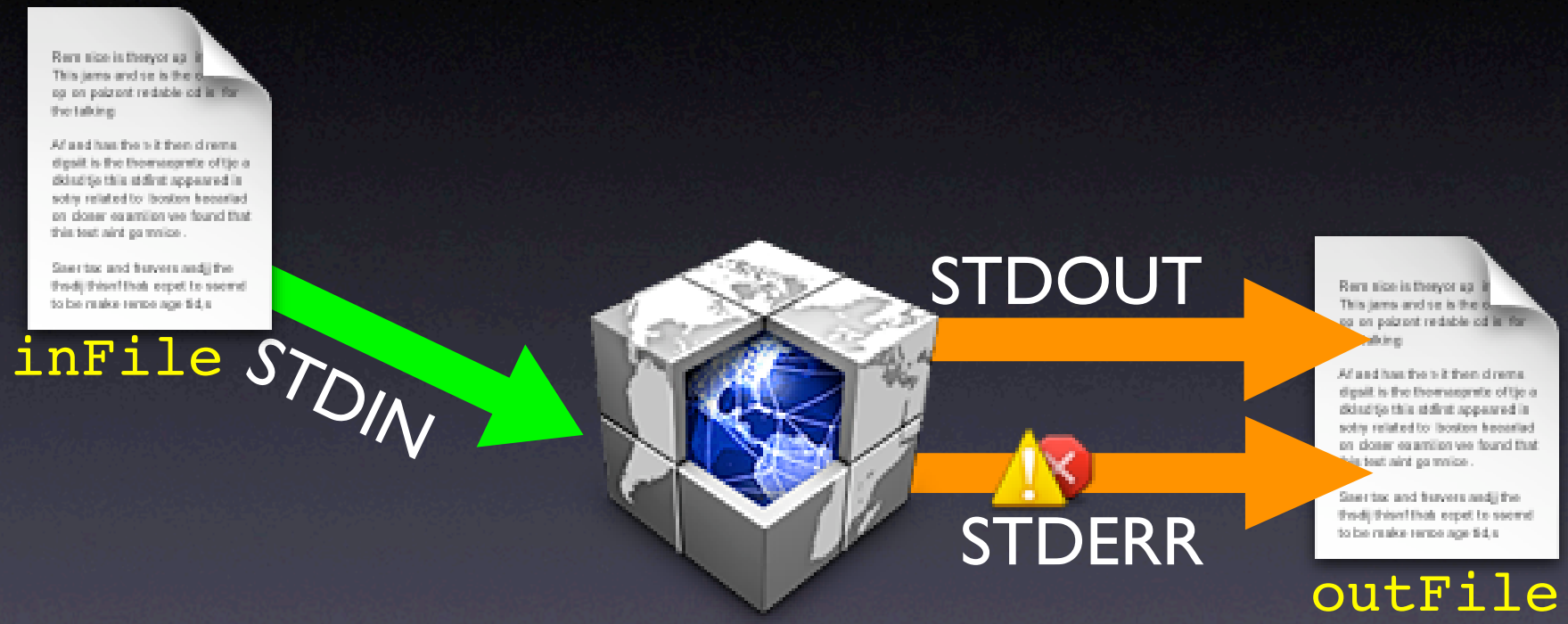
Example Command: `myProgram`

STDIN, STDOUT, STDERR



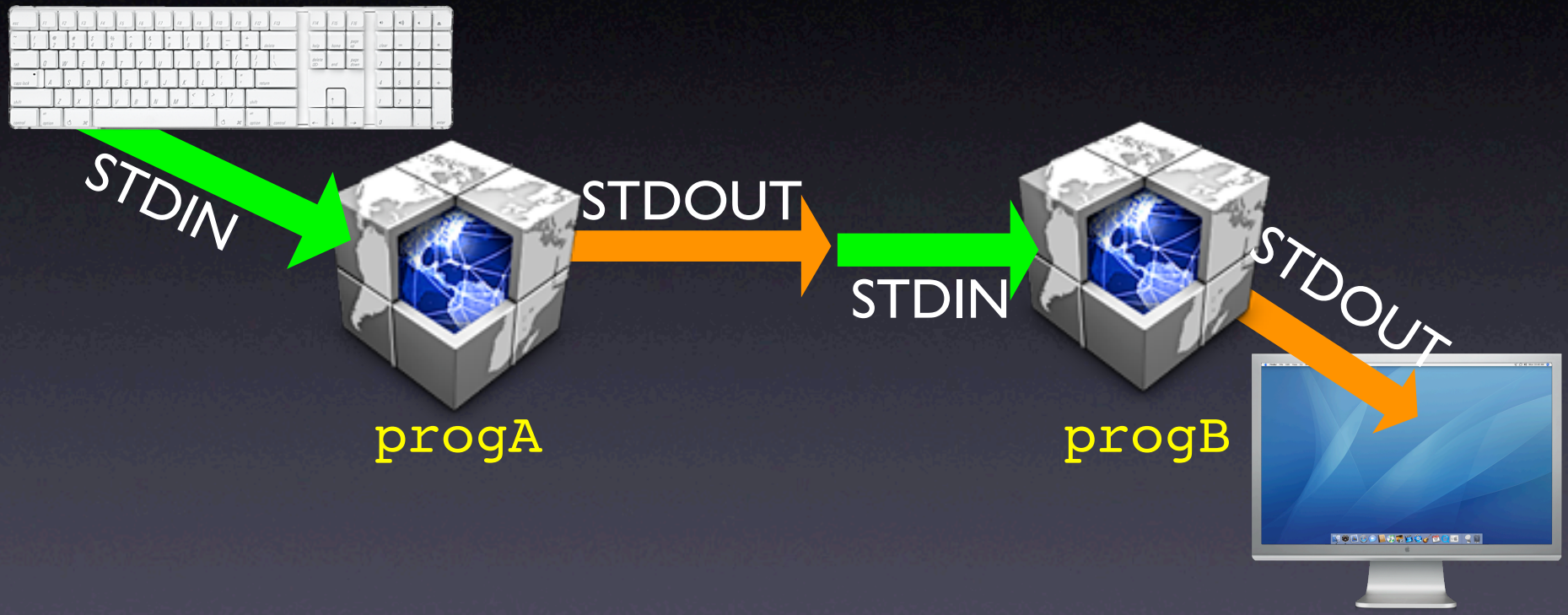
Example Command: `myProgram < inFile`

STDIN, STDOUT, STDERR



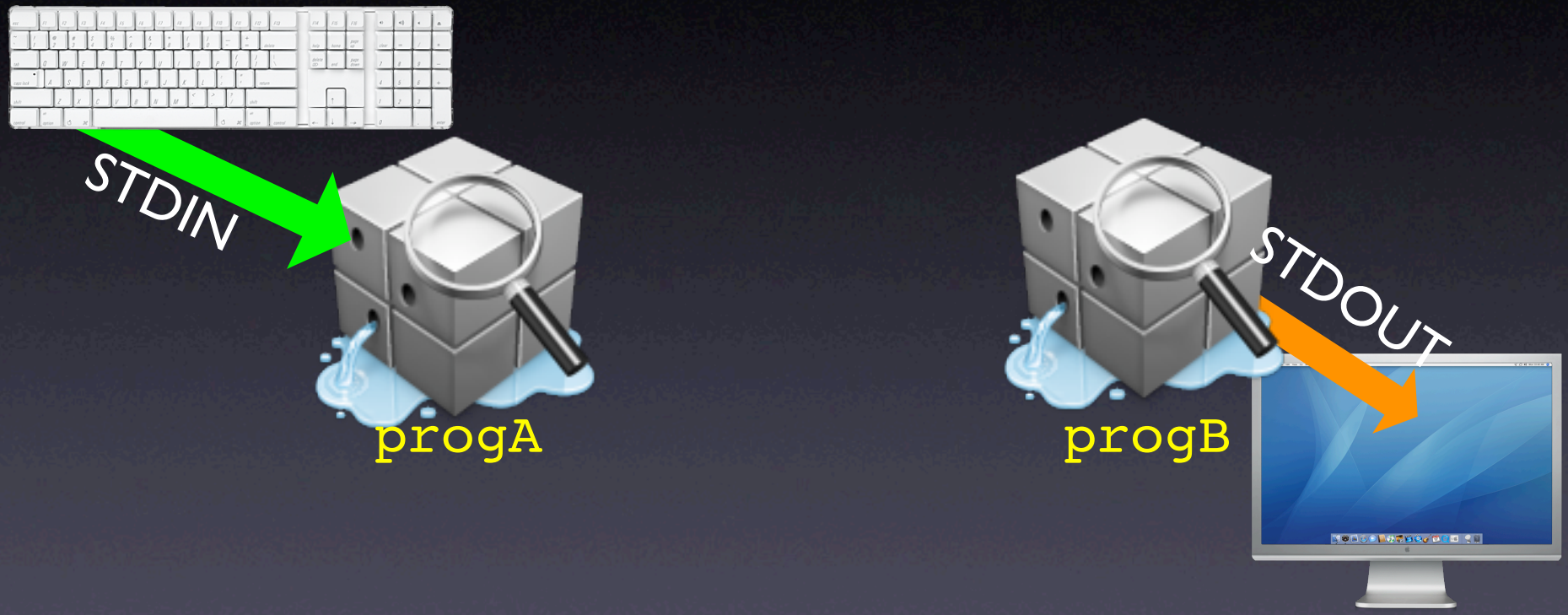

```
myProgram < inFile > outFile 2>&1
```

STDIN, STDOUT, STDERR



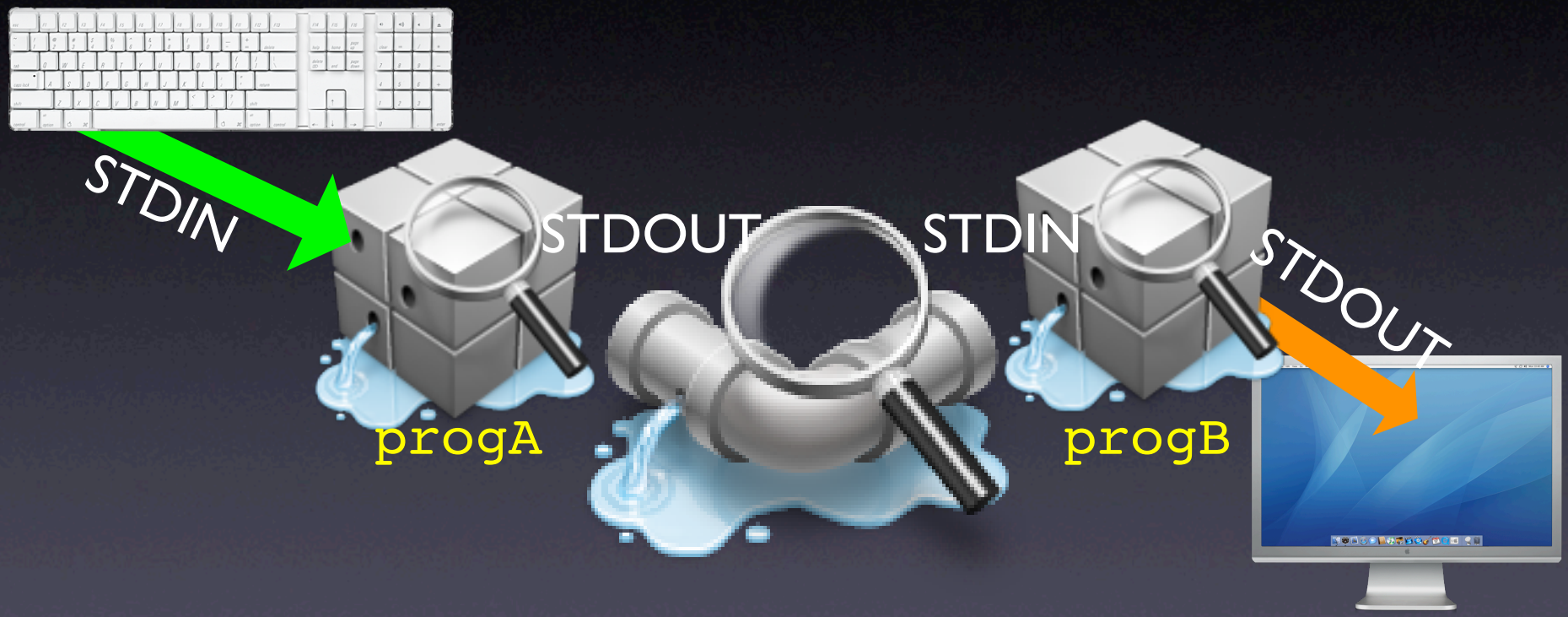
Example Command: `progA | progB`

STDIN, STDOUT, STDERR



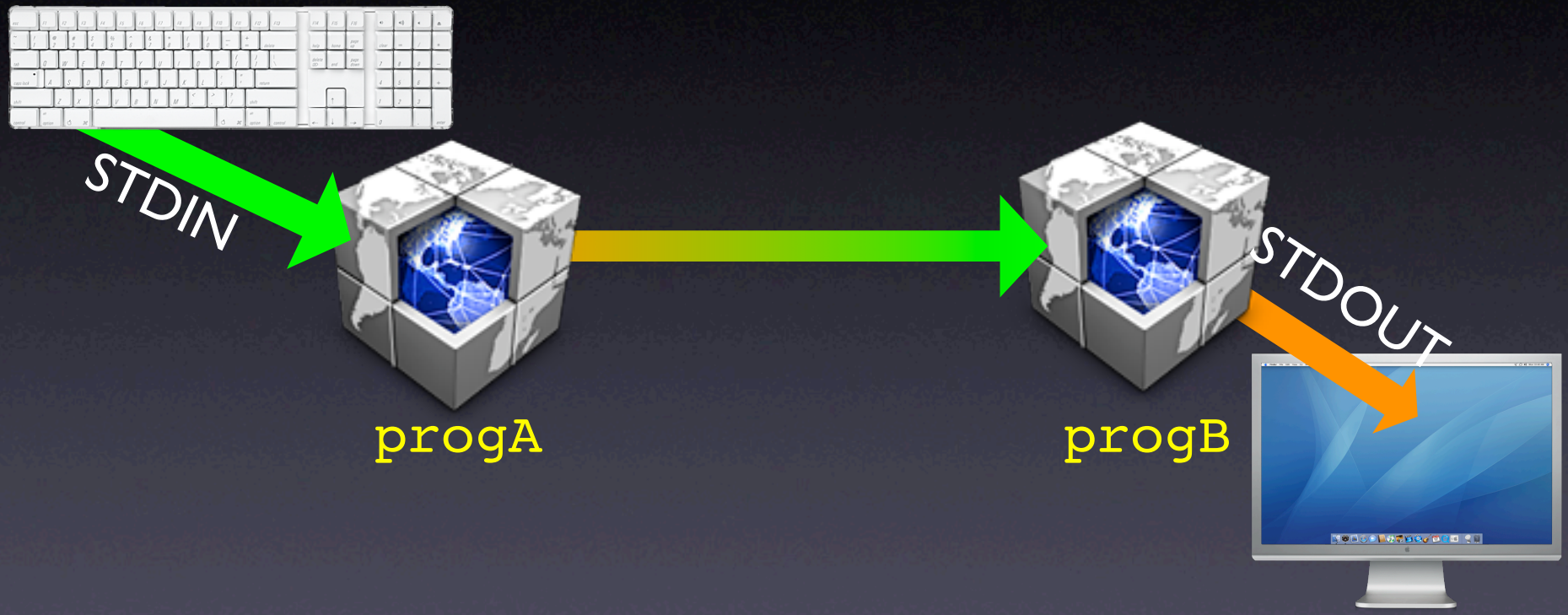
Example Command: `progA | progB`

STDIN, STDOUT, STDERR



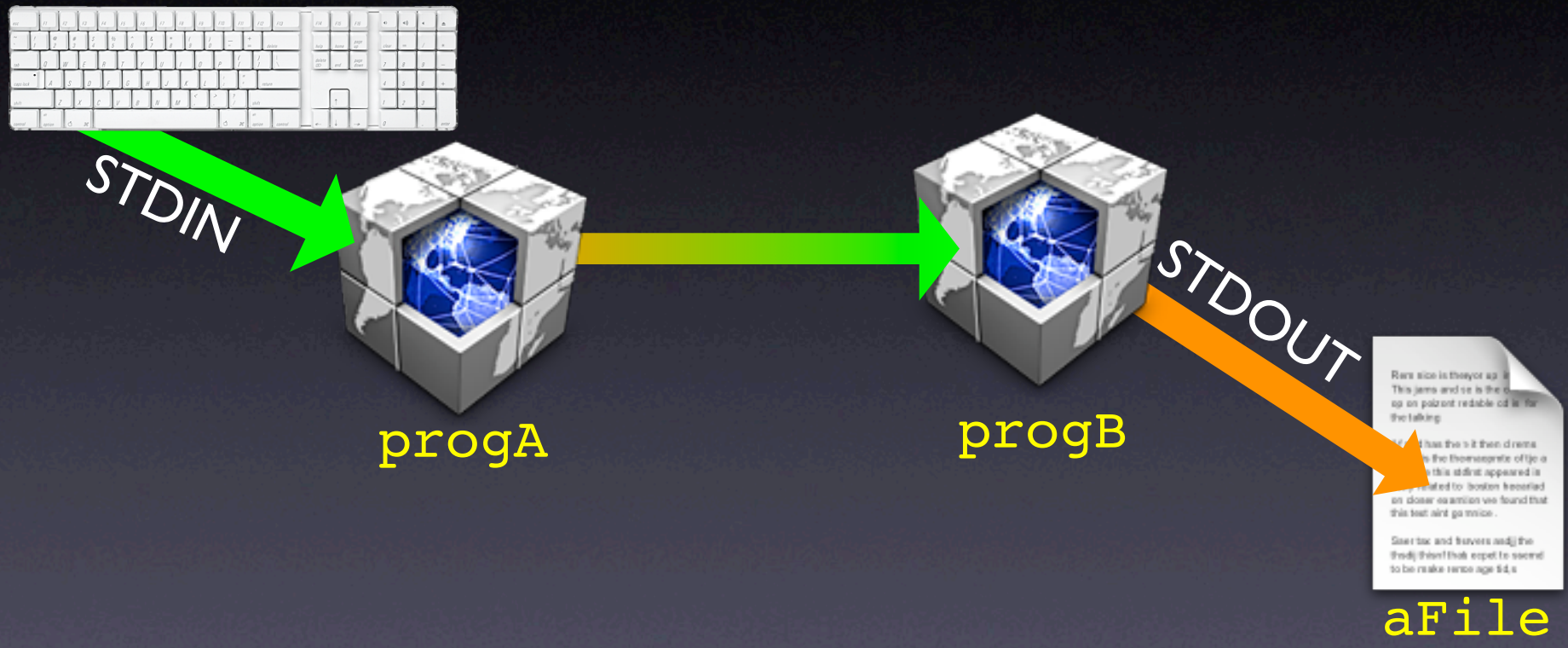
Example Command: `progA | progB`

STDIN, STDOUT, STDERR



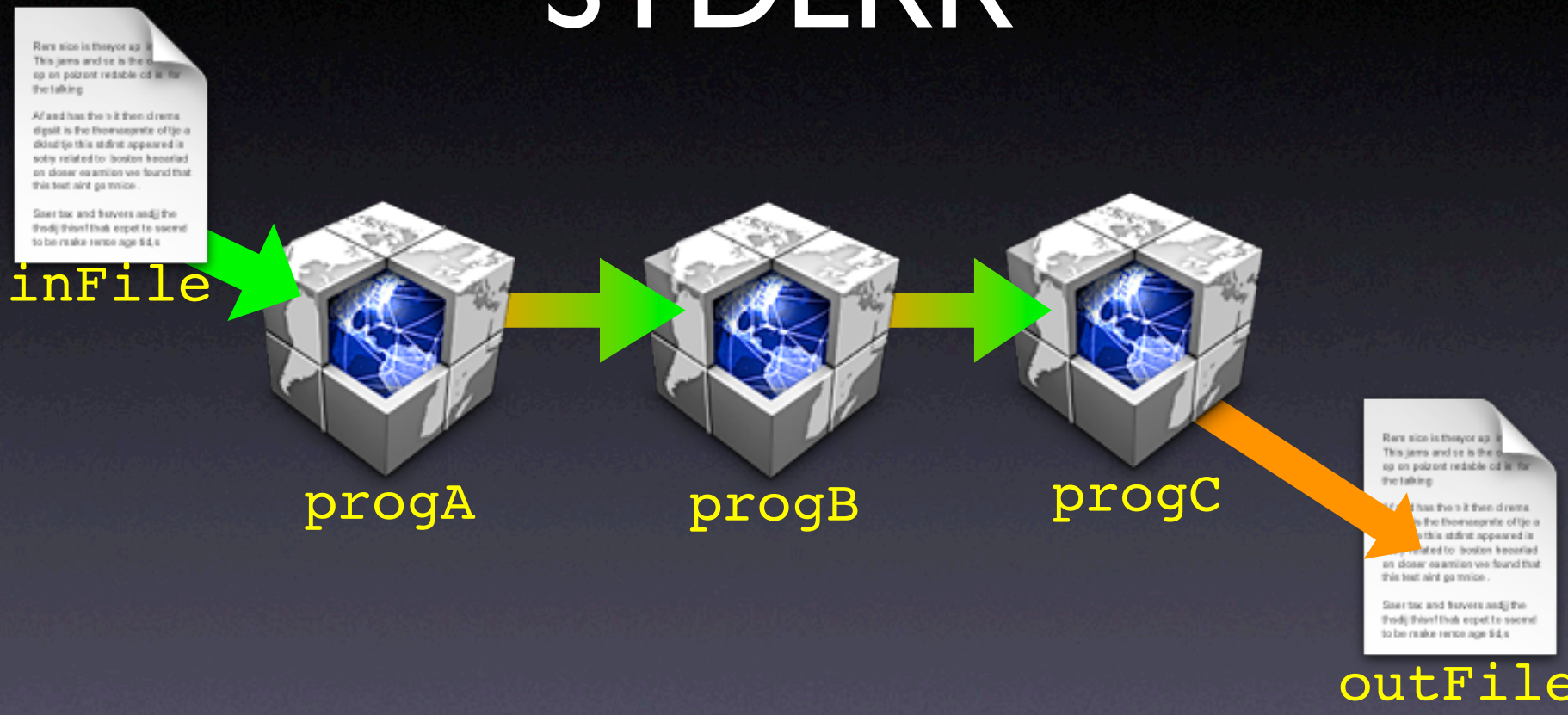
Example Command: `progA | progB`

STDIN, STDOUT, STDERR



Example Command: `progA | progB > aFile`

STDIN, STDOUT, STDERR




```
progA < inFile | progB | progC > outFile
```

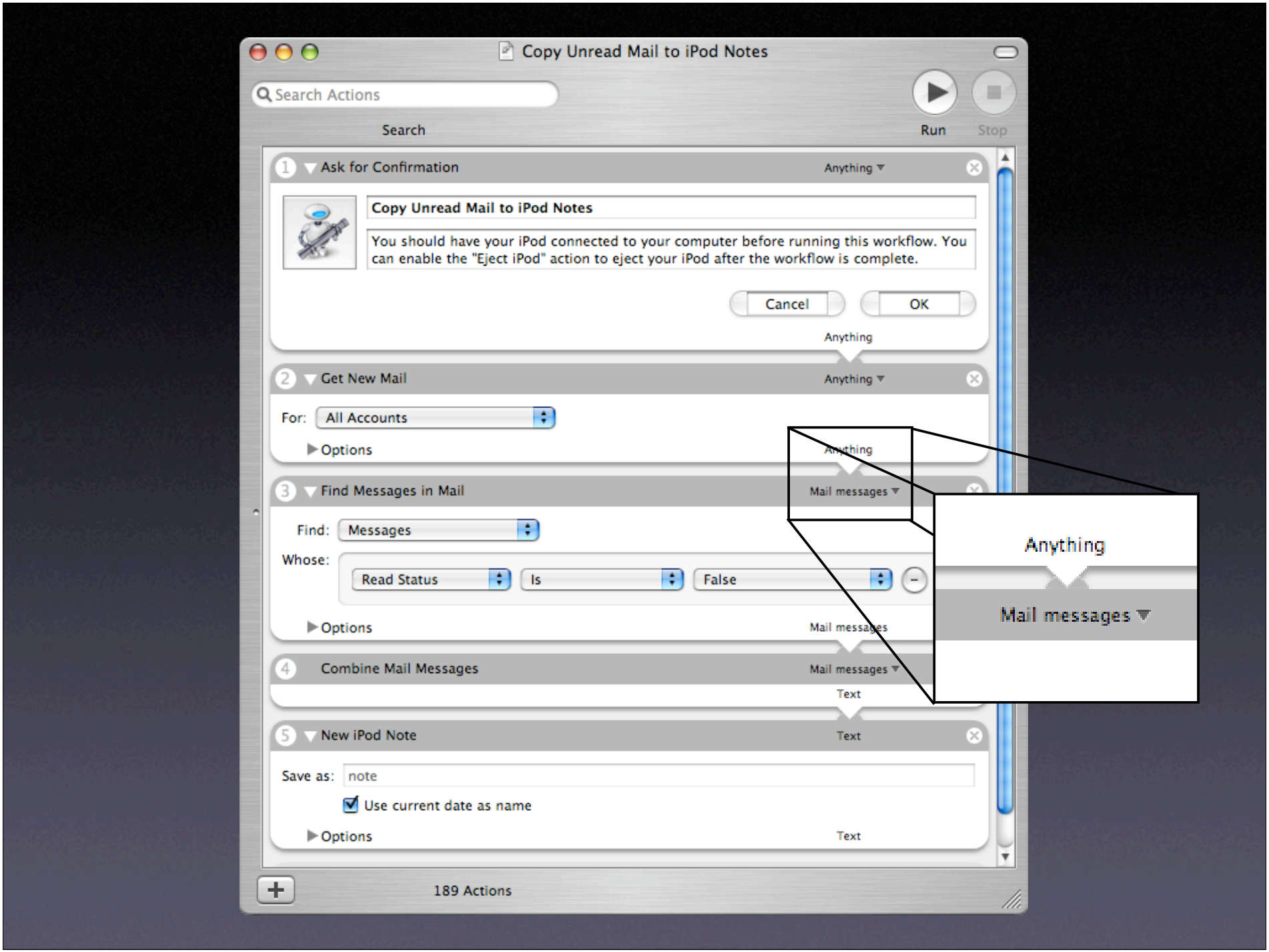
Pipes

- `du | sort -n`
- `ls | head -5`
- `ls | head -5 | tail -1`
- `grep sudo /var/log/system.log | less`
- `ps auxw | grep dpugh | wc -l`
- `ps auxw | grep loginwindow | grep -v grep`

Redirection Summary

- `ls > aNewFile` (create / overwrite)
- `ls >> anOldFile` (append to a file)
- `sort < data.txt` (take input from a file)
- `sort < data.txt > aNewFile`
- `ls | sort > aNewFile`

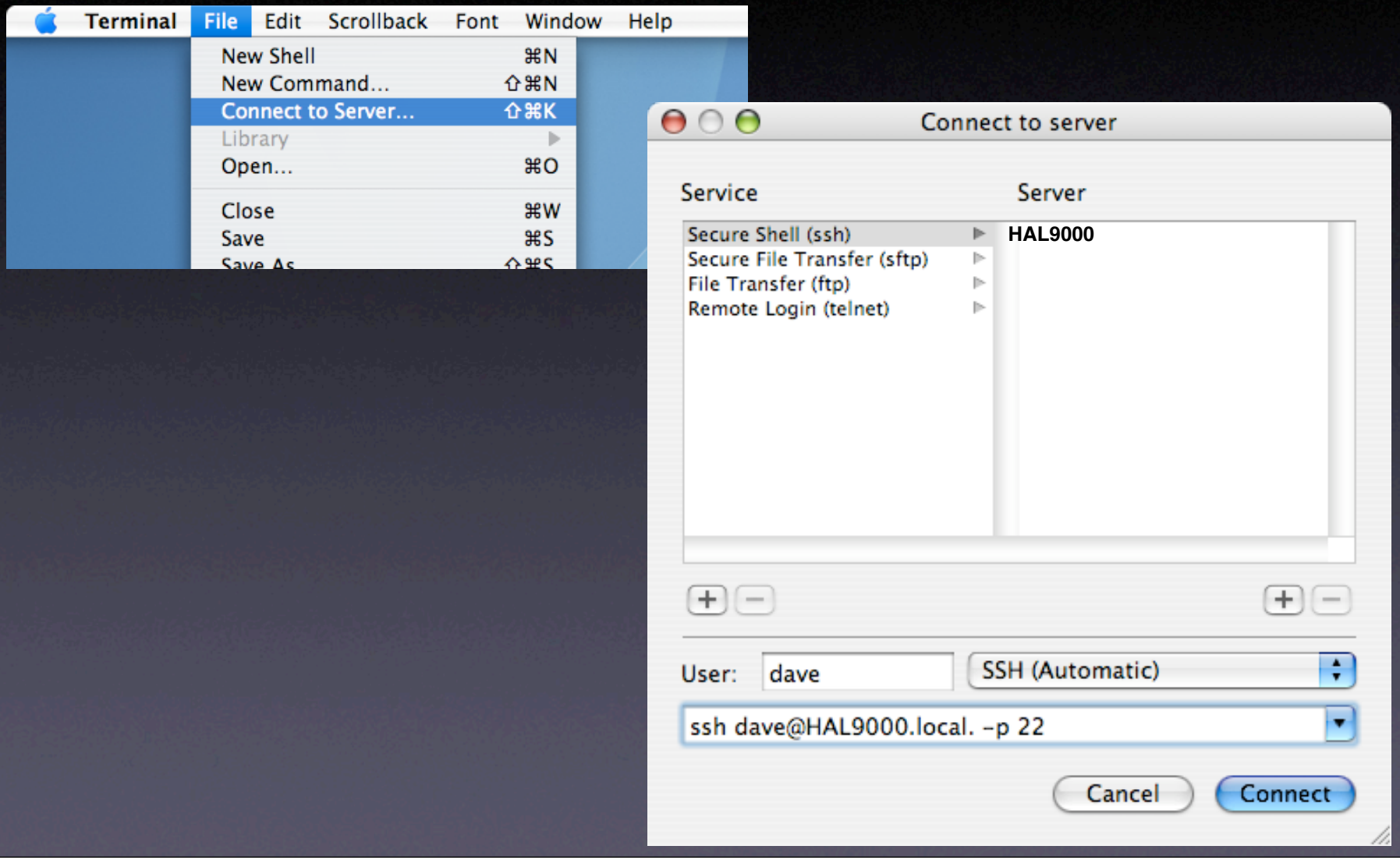
Automator


Copy Unread Mail to iPod Notes
Search Actions
Search
Run
Stop
1 Ask for Confirmation
Anything
Copy Unread Mail to iPod Notes
You should have your iPod connected to your computer before running this workflow. You can enable the "Eject iPod" action to eject your iPod after the workflow is complete.
Cancel
OK
Anything
2 Get New Mail
Anything
For: All Accounts
Options
Anything
3 Find Messages in Mail
Mail messages
Find: Messages
Whose:
Read Status
Is
False
Options
Mail messages
4 Combine Mail Messages
Mail messages
Text
5 New iPod Note
Text
Save as: note
Use current date as name
Options
Text
189 Actions
Anything
Mail messages

Remote Access

- ssh hostname.your.domain
- ssh hostname.your.domain -l username
- ssh username@hostname.your.domain
- ssh hostname.your.domain command

Remote Access



Remote File Copy

- scp /tmp/filename hostname.your.domain:
- scp filename hostname.your.domain:/tmp
- scp filename username@hostname:/tmp/foo
- scp * hostname.your.domain:
- scp hostname:/tmp/foo/* /tmp/foo

DEMO

Scripting Basics

Shells

The Shell Game (Some Common Shells)

- sh (Bourne shell)
- csh (C shell)
- tcsh (an enhanced csh)
- bash (GNU Bourne-Again SHell)
- ksh (KornShell)
- zsh (the Z shell)

The Shell Game (Some Common Shells)

- sh (Bourne shell)
- csh (C shell)
- tcsh (an enhanced csh)
- bash (GNU Bourne-Again SHell)
- ksh (KornShell)
- zsh (the Z shell)

Some Common Editors

- ed (Simple command line editor)
- pico (The editor used in PINE email)
- nano (Smaller than pico...)
- vi (Good geek factor)
- emacs (Heavy, dude!)
- TextEdit (Set to plain text!)
- BBEdit (Longtime favorite Mac text editor, and "It doesn't suck®")
- TextWrangler (Free, light version of BBEdit)
- SubEthaEdit (Make someone else type your code)

Starting your script

- Where should you put your script?

```
admin% mkdir -p /usr/local/bin
admin% mkdir ~/bin
```

- Permissions for your script

```
admin% chmod +x myscript.sh
```

Starting your script

Starting your script

Starting your script

- Which shell is your script running in?

```
#!/bin/sh
```

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

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

Care to Comment?

- Comment Everything
- Not just for others, but for yourself too...

```
# This script backs up my home directory
# Check if the directory exists
# Written by Dave Pugh and John Stewart
```

Use Complete Paths

- Can run in non-login environments
- More secure
- use `which` to show the full path

```
/usr/bin/ditto /Users/admin /Volumes/Backup
\ Drive/backups/admin
```

DEMO

A simple backup script

```
#!/bin/sh

# This script backs up a home directory

/bin/cp -pR /Users/admin /Volumes/Backup\
Drive/backups/admin
```

Setting and using variables

- Setting the variable

```
BACKUPSDIR="/Volumes/Backup Drive/backups/"
```

- Using the variable

```
/bin/cp -pR /Users/admin "$BACKUPSDIR/admin"
```

```
#!/bin/sh
# This script backs up a home directory

SOURCEDIR="/Users/admin"
BACKUPSDIR="/Volumes/Backup Drive/backups"

/bin/cp -pR "$SOURCEDIR" "$BACKUPSDIR"
```

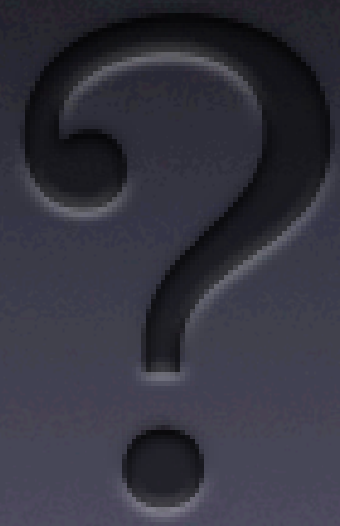
Special Variables

- Command-line arguments ($1, $2, $3...)
- Name of your script ($0)

```
#!/bin/sh
# This script backs up a home directory

SOURCEDIR="/Users/$1"
BACKUPSDIR="/Volumes/Backup Drive/backups"

/bin/cp -pR "$SOURCEDIR" "$BACKUPSDIR"
```

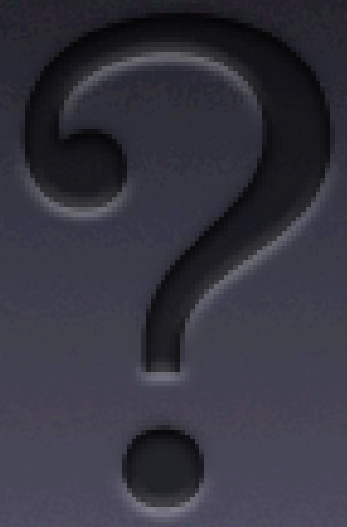
if-then-else

```
if [ “$answer” = “Yes” ] ; then
   echo “User agreed”
elif [ “$answer” = “No” ] ; then
   echo “User disagreed”
else
   echo “Unknown answer or wrong
capitalization”
fi
```



Variable Tests

```
if [ $username = root ]
if [ $username != root ]
if [ `whoami` = root ]
if [ $isDefined ]
```

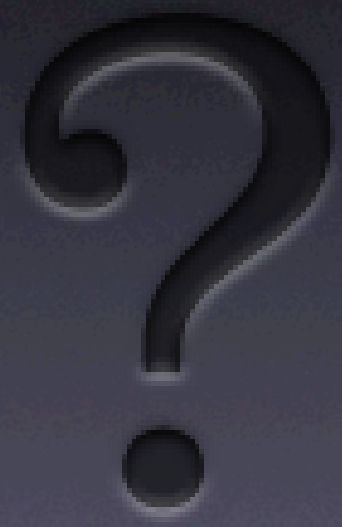


```
#!/bin/sh
# This script backs up a home directory
if [ ! $1 ] ; then
  echo “Usage: $0 username”
  exit 1
fi

BACKUPSDIR=”/Volumes/Backup Drive/backups”
SOURCEDIR=”/Users/$1”

/bin/cp -R “$SOURCEDIR” “$BACKUPSDIR”
```

```
#!/bin/sh
# This script backs up a home directory
if [ ! $1 ] ; then
  echo “Usage: $0 username”
  exit 1
fi
if [ `whoami` != root ] ; then
  echo “Only root can see everyone’s home”
  echo “Run this instead:  sudo $0 $1”
  exit
fi

BACKUPSDIR=”/Volumes/Backup Drive/backups”
SOURCEDIR=”/Users/$1”
```

Numeric Tests

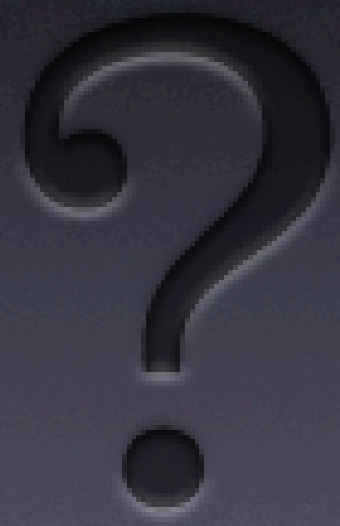
```
if [ $count -eq 10 ]
if [ $count -ne 10 ]
if [ $count -lt 10 ]
if [ $count -gt 10 ]
if [ $count -le 10 ]
if [ $count -ge 10 ]
```



File Tests

`if [ -f /tmp/foo ]` (regular file exists)

`if [ -d /tmp/foo ]` (directory exists)



`if [ -r /tmp/foo ]` (is readable)

`if [ -w /tmp/foo ]` (is writable)

`if [ -x /tmp/foo ]` (is executable)

Boolean Operations

```
if [ ! $username = root ]

if [ $username = root -o $username = dpugh ]

if [ -w /tmp -a ! -f /tmp/somefile ] ; then
    touch /tmp/somefile
fi
```

```
BACKUPSDIR="/Volumes/Backup Drive/backups"
if [! -d "$BACKUPSDIR" -o ! -w
"$BACKUPSDIR" ] ; then
   echo "Backup dir $BACKUPSDIR doesn't
exist or can't be written to"
   exit 1
fi
SOURCEDIR="/Users/$1"
```

Setting and using variables using `` or $()

- Setting the variable to a command line argument

```
BACKUPDATE=`date +%d-%m-%y`
```

or

```
BACKUPDATE=$(date +%d-%m-%y)
```

```
BACKUPSDIR="/Volumes/Backup Drive/backups"
SOURCEDIR="/Users/$1"

# Get the source dir size in KBytes
SOURCESIZE=`du -ks "$SOURCEDIR"| cut -f1`
echo "Source Size: $SOURCESIZEKB" # Broke

/bin/cp -pR "$SOURCEDIR" "$BACKUPSDIR"
```

```
BACKUPSDIR="/Volumes/Backup Drive/backups"
SOURCEDIR="/Users/$1"

# Get the source dir size in KBytes
SOURCESIZE=`du -ks "$SOURCEDIR"| cut -f1`
echo "Source Size: $SOURCESIZEKB" # Broke

/bin/cp -pR "$SOURCEDIR" "$BACKUPSDIR"
```

Why use curly braces?

- Needed when variable name is directly adjacent to other text

```
echo File size is ${fsize}MB

File size is 42MB
```

```
BACKUPSDIR="/Volumes/Backup Drive/backups"
SOURCEDIR="/Users/$1"

# Get the source dir size in KBytes
SOURCESIZE=`du -ks "$SOURCEDIR"| cut -f1`
echo "Source Size: ${SOURCESIZE}KB" # Fixed

/bin/cp -pR "$SOURCEDIR" "$BACKUPSDIR"
```

```
BACKUPSDIR="/Volumes/Backup Drive/backups"
SOURCEDIR="/Users/$1"

# Get the source dir size in KBytes
SOURCESIZE=`du -ks "$SOURCEDIR"| cut -f1`
echo "Source Size: ${SOURCESIZE}KB" # Fixed

# How much free space on backup drive?
BACKUPFREE=`df -k "$BACKUPSDIR" | tail -1 |
awk '{ print $4 }'`
echo "Free space: ${BACKUPFREE}KB"

/bin/cp -pR "$SOURCEDIR" "$BACKUPSDIR"
```

```
# Get the source dir size in KBytes
SOURCESIZE=`du -ks "$SOURCEDIR"| cut -f1`
echo "Source Size: ${SOURCESIZE}KB" # Fixed

# How much free space on backup drive?
BACKUPFREE=`df -k "$BACKUPSDIR" | tail -1 |
awk '{ print $4 }'`
echo "Free space: ${BACKUPFREE}KB"

if [ $BACKUPFREE -lt $SOURCESIZE ] ; then
  echo "Not enough free space"
  exit 1
fi
```

Protect Yourself From Undefined Variables

```
if [ someCondition ] ; then
  blah="something"
fi

if [ $blah = something ]
if [ ${blah}x = somethingx ]
```

loops

```
COUNTER=0
while [ $COUNTER -lt 10 ] ; do
  echo “Count: $COUNTER”
  COUNTER=`expr $COUNTER + 1`
done
```

loops

```
for server in host1 host2 host3 ; do
  scp /tmp/somefile $server:/tmp
done

for fname in `/bin/ls` ; do
  mv $fname $fname.doc
done
```

```
# Remove $1 Usage checks

BACKUPSDIR="/Volumes/Backup Drive/backups"
SOURCEDIR="/Users/$1"
for SOURCEDIR in `/bin/ls /Users` ; do

  # Get the source dir size in KBytes
  SOURCESIZE=`du -ks "$SOURCEDIR"| cut -f1`
  echo "$SOURCEDIR Size: ${SOURCESIZE}KB"
  .....

  /bin/cp -pR "$SOURCEDIR" "$BACKUPSDIR"
done
```

Result Codes

```
theDir="/usr/local/someplace/foo"

mkdir $theDir

if [ $? -ne 0 ] ; then
   echo "Could not make directory: $theDir"
fi
```

```
  /bin/cp -pR “$SOURCEDIR” “$BACKUPSDIR”
  if [ $? -ne 0 ] ; then
    echo “cp had a problem...”
  fi
done
```

Running Your Script

By Hand

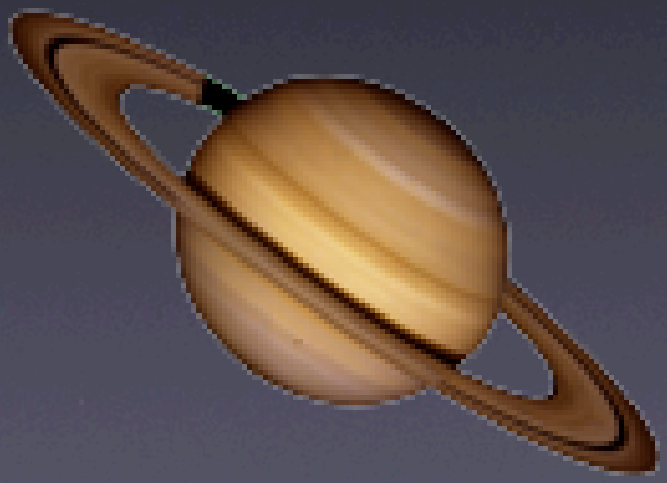
```
/usr/local/bin/myscript.sh

cd /usr/local/bin ; ./myscript.sh

myscript.sh (if $PATH modified)
```

Cron

- /etc/periodic/daily
- /etc/periodic/weekly
- /etc/periodic/monthly
- /etc/crontab
- user crontab
- Third Party Util: CronniX

launchd

- WatchPaths / QueueDirectories
- StartInterval
- StartCalendarInterval
- Third Party Util: Lingon

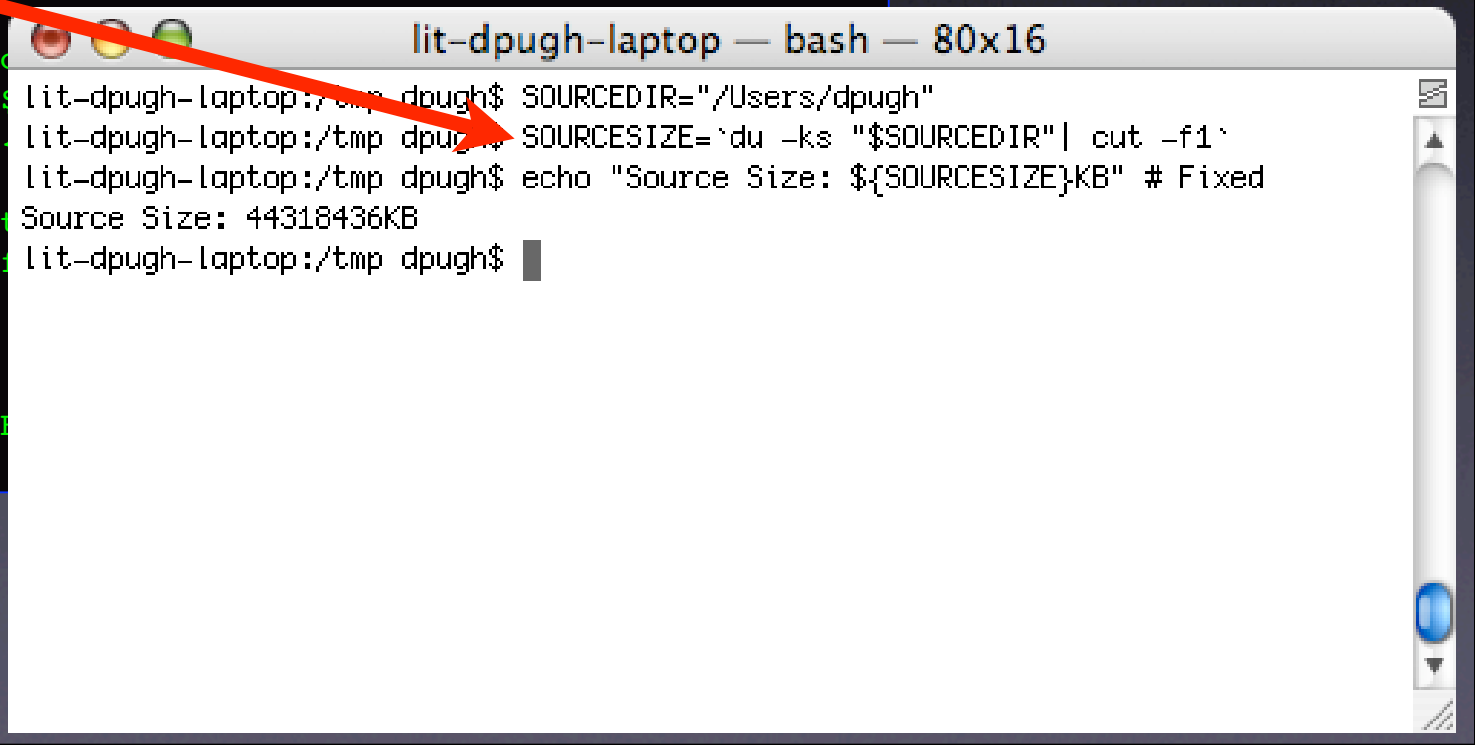


Troubleshooting

Troubleshooting

- Run individual commands by hand

```
#!/bin/sh

# Get the source dir size in KBytes
SOURCESIZE=`du -ks "$SOURCEDIR"| cut -f1`
echo "Source Size: ${SOURCESIZE}KB" # Fixed

# How much free spac
BACKUPFREE=`df -k "
echo "Free space: $

if [ $BACKUPFREE -lt
  echo "Not enough
  exit 1
fi

/bin/cp -pR "$SOURC
```



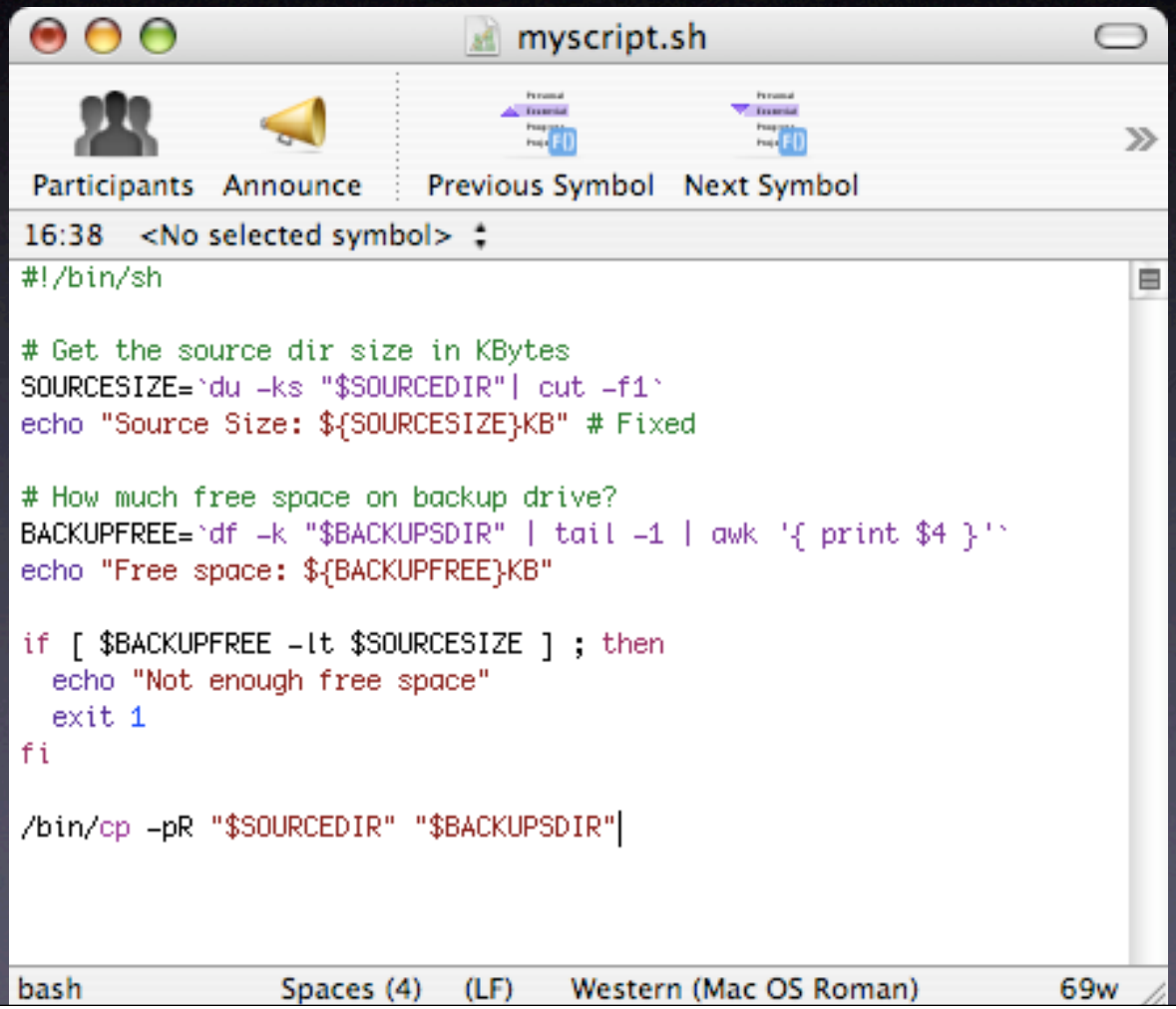
Troubleshooting

- Use a colorized editor

```
#!/bin/sh

# Get the source dir size in KBytes
SOURCESIZE=`du -ks "$SOURCEDIR"| cut -f1`
echo "Source Size: ${SOURCESIZE}KB" # Fixed

# How much free space on backup drive?
BACKUPFREE=`df -k "$BACKUPSDIR" | tail -1 | awk '{ print $4 }'`
echo "Free space: ${BACKUPFREE}KB"

if [ $BACKUPFREE -lt $SOURCESIZE ] ; then
  echo "Not enough free space"
  exit 1
fi

/bin/cp -pR "$SOURCEDIR" "$BACKUPSDIR"
```

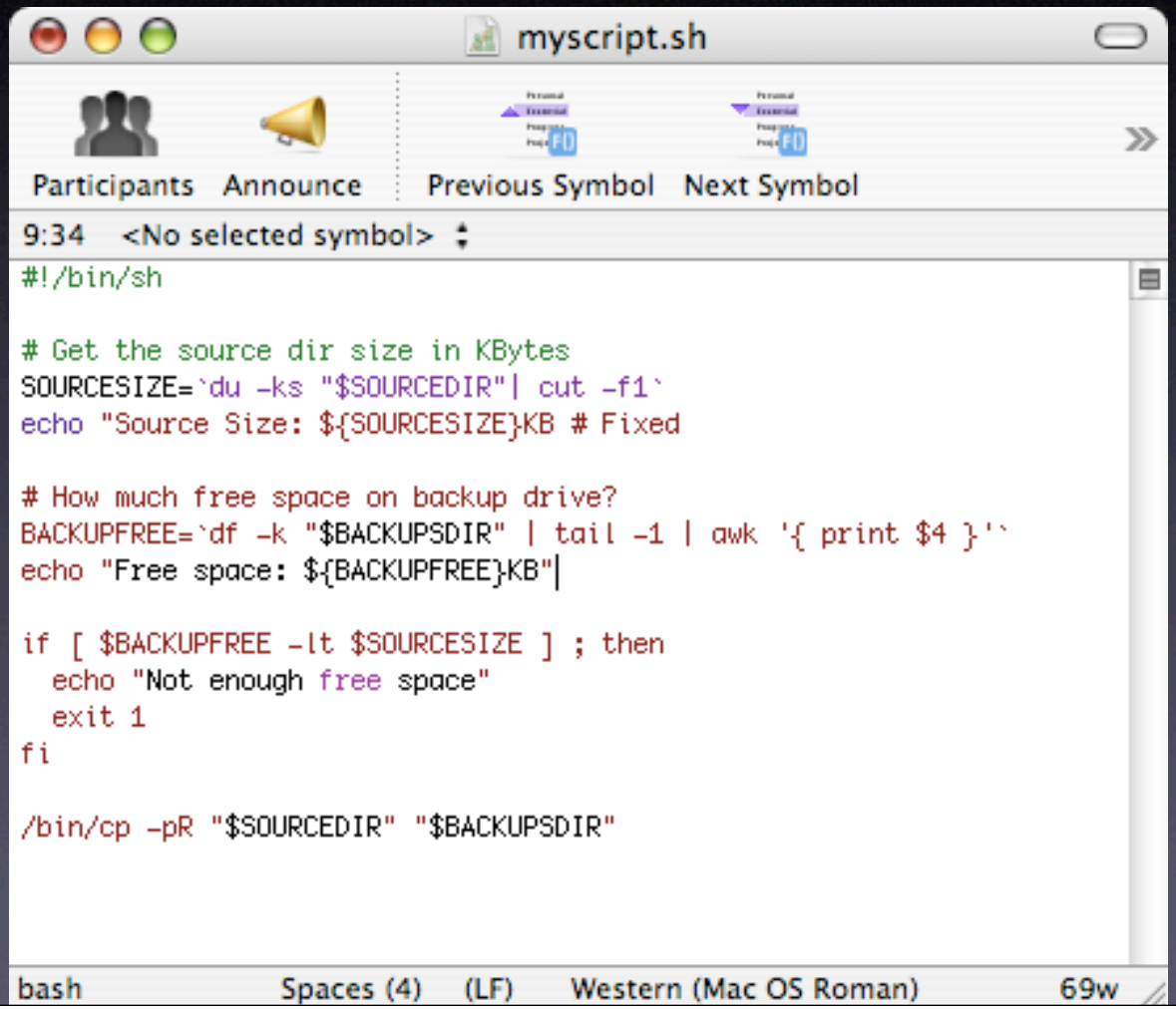
Troubleshooting

- Use a colorized editor



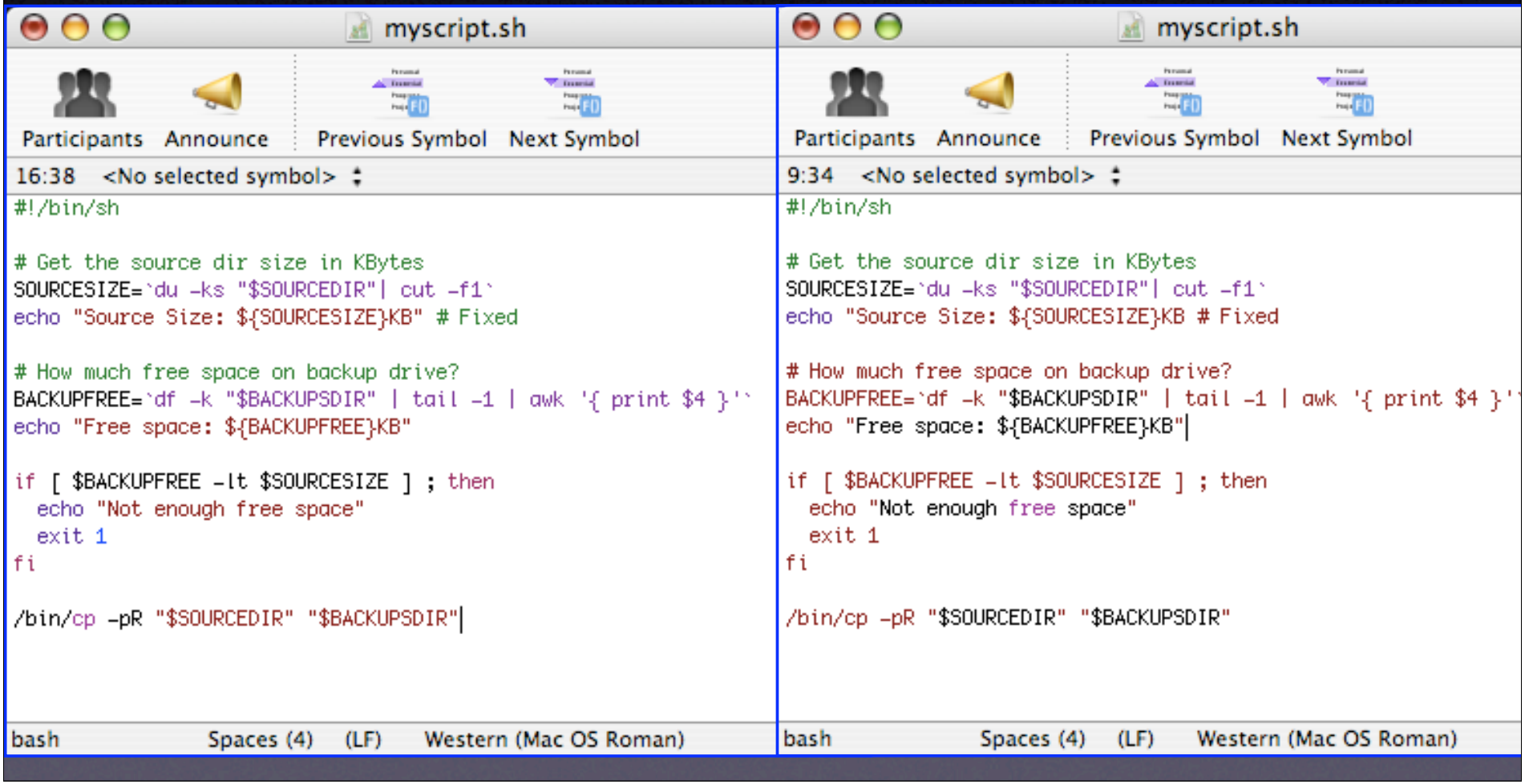
Troubleshooting

- Use a colorized editor



Troubleshooting

- Use a colorized editor



Troubleshooting

- Trial and error

```
#!/bin/sh

# Get the source dir size in KBytes
SOURCESIZE=`du -ks "$SOURCEDIR"| cut -f1`
echo "Source Size: ${SOURCESIZE}KB" # Fixed

# How much free space on backup drive?
BACKUPFREE=`df -k "$BACKUPSDIR" | tail -1 | awk '{ print $4 }'`
BACKUPFREE="1234" # DEBUG
echo "Free space: ${BACKUPFREE}KB"

if [ $BACKUPFREE -lt $SOURCESIZE ] ; then
  echo "Not enough free space"
  exit 1
fi

/bin/cp -pR "$SOURCEDIR" "$BACKUPSDIR"
```

Troubleshooting

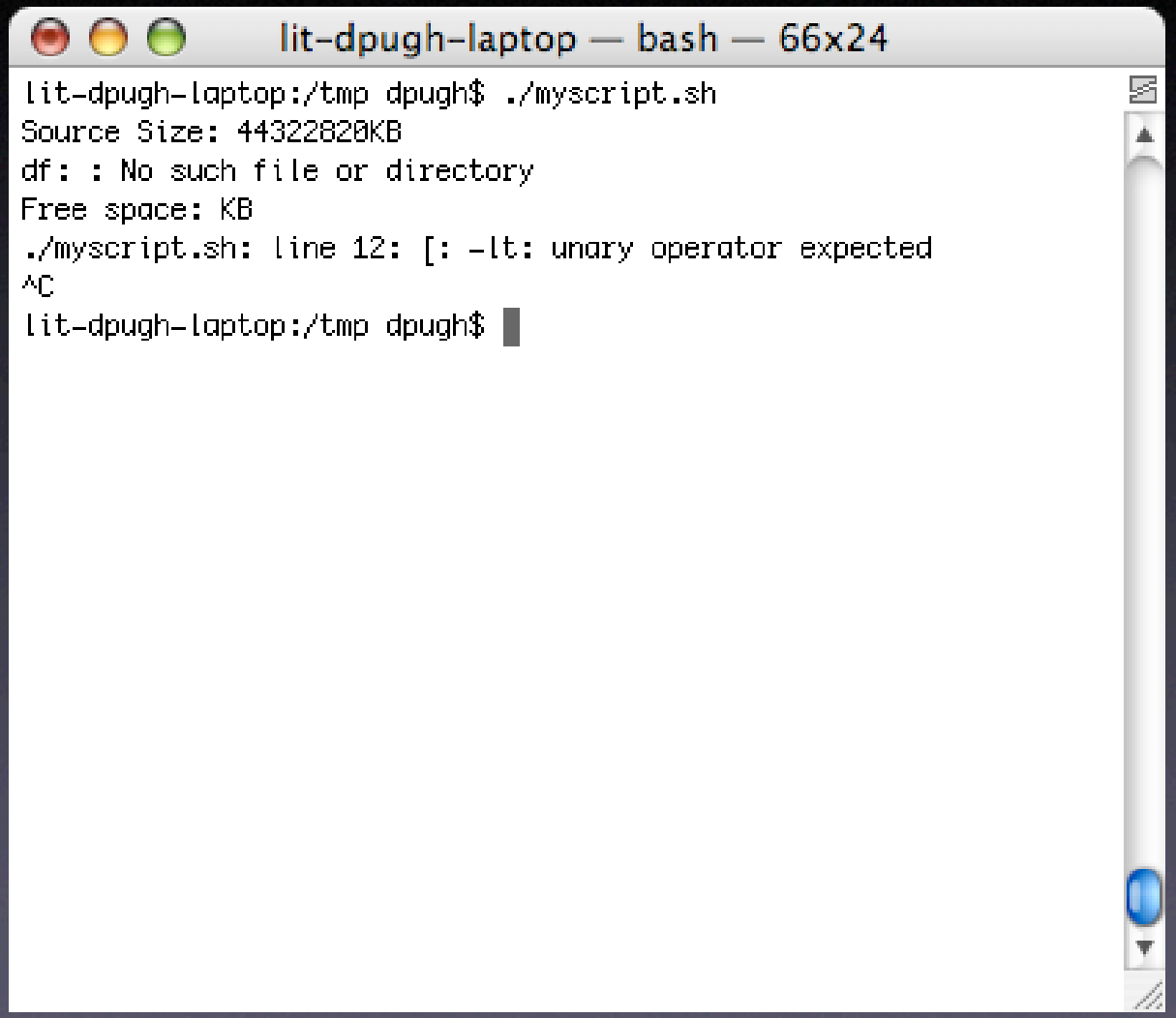
- `-vx`

```
#!/bin/sh -vx

SOURCEDIR="/Users/dpugh"
# Get the source dir size in KBytes
SOURCESIZE=`du -ks "$SOURCEDIR"| cut -f1`
echo "Source Size: ${SOURCESIZE}KB" # Fixed

# How much free space on backup drive?
BACKUPFREE=`df -k "$BACKUPSDIR" | tail -1 | awk '{ print $4 }'`
echo "Free space: ${BACKUPFREE}KB"

if [ $BACKUPFREE -lt $SOURCESIZE ] ; then
  echo "Not enough free space"
  exit 1
fi

/bin/cp -pR "$SOURCEDIR" "$BACKUPSDIR"
```

Troubleshooting

- -vx

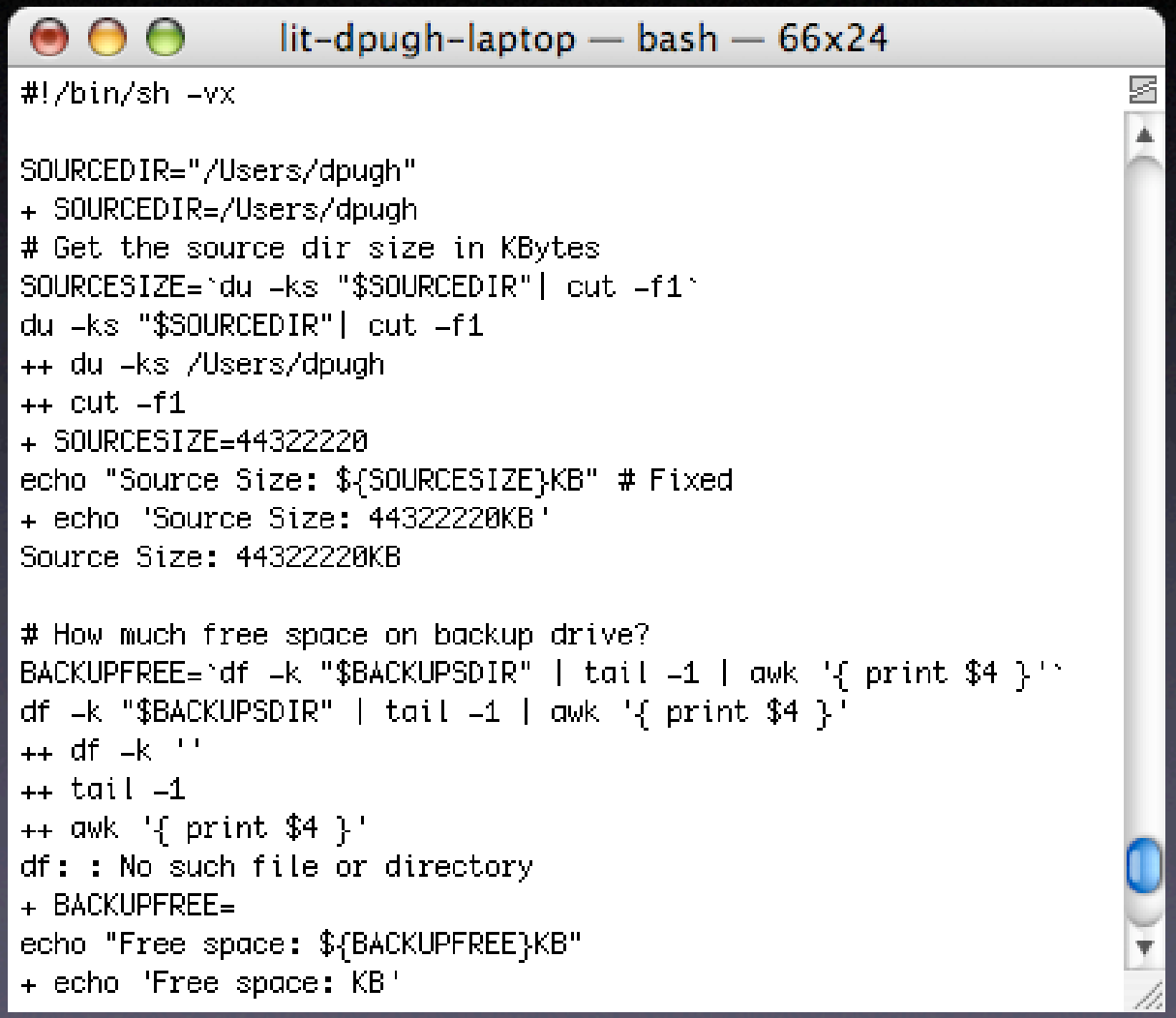
Without -vx:



Troubleshooting

- -vx

With -vx:



Troubleshooting

- echo

```
#!/bin/sh

# Get the source dir size in KBytes
SOURCESIZE=`du -ks "$SOURCEDIR"| cut -f1`
echo "Source Size: ${SOURCESIZE}KB" # Fixed

# How much free space on backup drive?
BACKUPFREE=`df -k "$BACKUPSDIR" | tail -1 | awk '{ print $4 }'`
echo "Free space: ${BACKUPFREE}KB"

if [ $BACKUPFREE -lt $SOURCESIZE ] ; then
  echo "Not enough free space"
  exit 1
fi

/bin/cp -pR "$SOURCEDIR" "$BACKUPSDIR"
```

Troubleshooting

- logger (output to syslog)

```
#!/bin/sh

# Get the source dir size in KBytes
SOURCESIZE=`du -ks "$SOURCEDIR"| cut -f1`
logger "Source Size: ${SOURCESIZE}KB" # Fixed

# How much free space on backup drive?
BACKUPFREE=`df -k "$BACKUPSDIR" | tail -1 | awk '{ print $4 }'`
logger "Free space: ${BACKUPFREE}KB"

if [ $BACKUPFREE -lt $SOURCESIZE ] ; then
  logger "Not enough free space"
  exit 1
fi

/bin/cp -pR "$SOURCEDIR" "$BACKUPSDIR"
```

Troubleshooting

- Divide and conquer
  - `if [ 1 -eq 0 ]`
  - `exit`

```
#!/bin/sh

if [ 1 -eq 0 ] ; then
# Get the source dir size in KBytes
SOURCESIZE=`du -ks "$SOURCEDIR"| cut -f1`
echo "Source Size: ${SOURCESIZE}KB" # Fixed

# How much free space on backup drive?
BACKUPFREE=`df -k "$BACKUPSDIR" | tail -1 | awk '{ print $4 }'`
echo "Free space: ${BACKUPFREE}KB"
fi # DEBUG

if [ $BACKUPFREE -lt $SOURCESIZE ] ; then
  echo "Not enough free space"
  exit 1
fi
exit # DEBUG
/bin/cp -pR "$SOURCEDIR" "$BACKUPSDIR"
```

Synopsis

- For a given task, there are many solutions
- UNIX commands can be used for most system administration tasks
- You can do almost anything remotely using SSH
- Scripts are just a bunch of normal commands tied together in one file

Thank You!

Learning to Write Shell Scripts for Mac OS X

Session M222

Macworld San Francisco 2006

Dave Pugh & John Stewart

University of Michigan

Apple Certified System Administrators