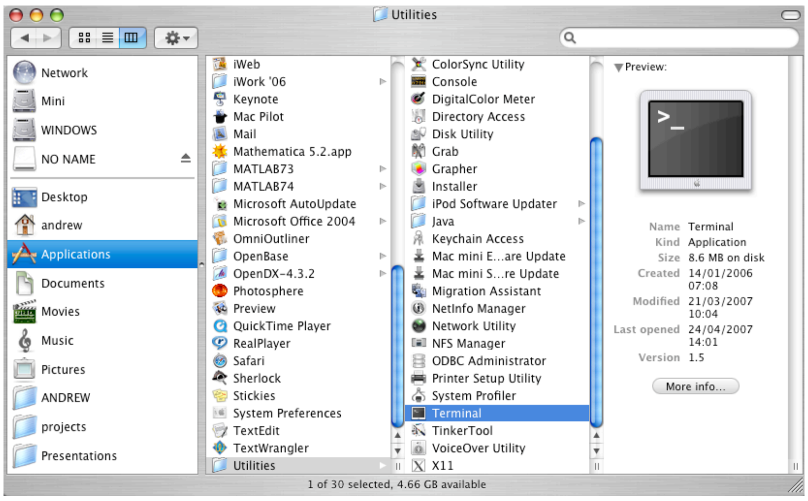
UNIX and RADIANCE

Giulio Antonutto & Andrew McNeil

Unix Terminal

Starting the Terminal Application

On a Mac the terminal program can be found in the /Applications/Utilities folder. When launched a window titled “Terminal - Bash - 80 x 24” will appear. This window contains a Unix prompt (which is in some ways similar to a DOS prompt for those familiar with windows). At the prompt you can type commands that will be executed within the terminal window.



Useful Unix Commands

The following UNIX commands will allow the user to navigate through the file system, manipulate files, check and terminate running processes.

Command	Description	Usage	Useful options
ls	list directory contents	**ls** **ls folder/**	**-l** lists additional file info **-a** lists all files (including hidden files)
cd	change directory	**cd folder/folder2/** **cd ..**	
more	view a text file one page at a time	**more file.txt**	
cp	copy file	**cp file.txt newfile.txt**	
mv	move a file	**mv oldloc.txt newloc..txt**	
rm	remove a file	**rm file**	DANGER, DO NOT USE: **rm ***
ps	list currently running processes	**ps**	
top	list the currently running processes and various system statistic	**top -u -s 5 5**	

Command	Description	Usage	Useful options
kill	kills a running process	**kill pid**	pid is the process id number which can be found by using ps.
open	open the current folder or a file	**open .**	use **-a** to specify the application to use
man	open manual page for a command	**man command**	to quit press esc and then q. to browse use arrows

These are just a few useful Unix commands. They operate in a standard manner across most flavors of UNIX.

Try typing **ls** at your UNIX prompt. It lists the contents of your current directory. Now type **ls -l** it displays more information of the contents in your current directory. Want to know more about the command? Type **man ls** to view the manual page for ls. 'man' will look for the manual page for any command. Most UNIX commands and Radiance programs have manual pages. if you're ever stuck give man a try!

Wildcard Characters

Character	Description
*	Represents any character(s) including none. Typing 'ls d*.txt' would list d.txt, d4.txt and dd_car.txt.
?	Represents any single character. Typing ls d?.txt would return d4.txt, but not d.txt nor dd_car.txt.

Directory Characters

Character	Description
.	Current directory
..	Parent directory (the directory up one level - containing the current directory)

Process control

Character	Description
ctrl - c	kills a process running in the foreground
ctrl - z	stops (pauses) a process running in the foreground
bg	continues a stopped (paused) process in the background
fg	returns a stopped (paused) or background process to the foreground

Pipes and Redirection

By default most commands or programs output data to what is known as the standard output. Basically, this means it displays it in the terminal. However, there will be occasions when we would prefer that the results of a command to be put in a file for use later. This can be achieved by using pipes and redirection.

For example, if you type **ls -l**, the directory contents will be listed in the terminal. If you type **ls -l > contents.txt**, the directory contents will be put in the file contents.txt. The '>' character is a redirection operator that tells the shell to put the output in a file. Type **more contents.txt** to view the contents of the file.

Shell Redirection Operators

Symbol	Usage	Description
>	cmd > file	Send the output of cmd to file. If the file exists, it will be overwritten.
>>	cmd >> file	Append the output of cmd to file. If the file exists, the output will be added to the end of the file.
<	cmd < file	Take input from file for use by cmd.
|	cmd1 | cmd2	Send the output from cmd1 to input of cmd2.

The '|' character is called a pipe. It is used to connect the output of one command with the input of another. For example type **ps -aux** at the prompt. Whoa, did you get all that? This command prints all the current processes running on the computer in order of processor usage. Now try **ps -aux | more** and use the enter key to scroll. The output of ps is sent to the program more which displays the data one page at a time.

Pipes and redirection are used often in Radiance. It is a convenient and efficient way of sending data to a file or to another program. For example, the following command will calculate the RGB spectral irradiance at each point in a grid file, converts to photopic illuminance and writes the results to a file in the format X Y Z Illuminance.

rtrace -h- -I -ab 2 -oov model.oct **<** grid.pts **|** rcalc -e '$1=$1;$2=$2;$3=$3;$4=$4*47.4+$5*119.9+$6*11.6' **>** E_grid.out

The inter-workings of this command are not important at the moment. Just understand that the file grid.pts given as input to the program rtrace using the '<' operator, the results are passed from rtrace to rcalc using '|' and the final data is written to the file E_grid.out using the '>' character.

Makefile

In the UNIX environment there is a utility called **make** that manages file dependencies and commands for creating files. The utility is intended for generating executable programs and rebuilding the components whose source codes and referenced libraries have changed since last compiled. However, make can also be used for managing radiance workflow. It can be used to manage interdependencies between model components and simulation data. It has the additional benefit of providing a comprehensive record of commands and options used to produce output.

All the commands required are put in a file called **Makefile**.

The best way to understand make is through example. The basic syntax for a make file is as follows:

```
product.out: depend1.txt depend2.inp
     command -option < depend1.txt| command2 < depend2.inp> file.out
```

In this example file.out is the object that is to be made. It relies on the files depend1.txt and depend2.inp. This is to say that if they don't exist file.out can not be made and if they have been updated file.out needs to be remade. The indented line following is the instruction for making file.out. To make file.out, at the prompt one would type **make product.out**. Make would then check to see if product.out existed. If it did, it would check the creation time of file.out and compare it with the modification time of the dependencies. If the dependencies have changed since product.out was made, it would make fproduct.out again (by executing the command on the line that follows). If product.out did not exist, make would make it.

Let's look at a radiance specific example:

```
# Makefile for classroom project

GEOMETRY = floor.rad walls.rad ceiling.rad glass.rad

model.oct: materials.rad $(GEOMETRY)
     oconv materials.rad $(GEOMETRY) > model.oct
```

```
model_jc.oct: model.oct sky_jc.rad
     oconv -i model.oct sky_jc.rad > model_jc.oct
     rm model_jc.amb

view1.raw: v1.vf model_jc.oct
     rpict -vf v1.vf -ab 2 -af model_jc.amb model_df.oct > view1.raw
```

This example may not be clear until after you are more familiar with radiance, but it may prove to be a useful reference later. Do remember to return to it as you get further into your analysis.

The first line is a comment, make will ignore lines that start with the character #, so use this to enter comments.

Next, the statement 'GEOMETRY = floor.rad walls...' is a variable declaration. Later in the file the variable is used in the making of model.oct. make will replace the syntax $(GEOMETRY) with "floor.rad walls.rad ceiling.rad glass.rad"

Let's suppose that you have run everything and are looking at the image view1.raw. You notice that the sky is sunny, but you inteded it to be overcast. So you need to change the file sky_jc.rad. Then instead of figuring out what needs to change, you simply type **make view1.raw**. Then make will check the dependencies it will see that it is dependent on the file model_jc.oct, so it checks the dependencies of model_jc.oct and sees that it is dependent on sky_jc.rad which has been modified since view1.raw was last made. Make then executes the commands that generate model_jc.oct (one of which deletes the ambient file since it is now invalid) it will then start to make view1.raw with the new model_jc.oct file.

X11 - X Window Server

Launching X11

Radiance uses an X window protocol to display graphical information. This approach is common with unix. Apple has provides an X window system with OSX. The X window system needs to be running for any of Radiance's graphical programs to run (rvu, ximage & objview).

To start X11 simply go to Applications/Utilities and double click the X11 icon. An xterm window will appear, this window can be hidden or minimized without causing disruption.

The First Time

Prior to using X11 with radiance from the terminal you will need to set the Display Variable in you bash profile. This only needs to be done once per user. To do this you need to add the following to the file .bash_profile in your home directory:

```
if [ -z "${DISPLAY}" ];then echo -n
export DISPLAY=':0'
fi
```

The file .bash_profile is a hidden file. This is indicated by the preceding period in the file name. Hidden files will not appear in the finder. The best way to access .bash_profile is by typing:

```
open -a textedit .bash_profile
```

BE CAREFUL! The file .bash_profile contains some important stuff and may cause a few things to stop working if you mess with it. Once this has been added and saved, quit and restart the terminal.

SCRIPTING

HOW TO START

Simply write always your commands sequentially into a text file within the radiance folder.

Use a lot of comments and provide reference for others.

comments may be inserted by starting a line with the **#** symbol like this:

```
#comments ... comments ... comments
```

To run such a list of commands, as a script just type:

```
sh command_file_name
```

HOW TO CONTINUE

If you are interested in developing further you may need to know how to create a command.

If is essentially like a script but with special permission and special header, follow this example:

Create the file 'run'

```
#/bin/tcsh -f

#the line above is required in all the command files

#now you may use any command radiance offer or special while, for cycles:
foreach view (1 2 3 4 5 6 7)

#note that variables are called with $
set settings = (-ab 4 -aa .2 -ad 1024 -as 512 -af model.amb -ps 1)

rpict -vf views/$view.vf $settings model.oct > $view.pic
pcond -h $view.pic > $view.pic.2
ra_tiff $view.pic.2 $view.tif

end
```

Give the proper permissions

Just type (run is the name of the file, it may be different):

```
chmod u+x run
```

Run the new tool

Just type (run is the name of the file, it may be different):

```
./run
```

Note: The script name is preceded by **./** which tells the terminal to find the script in the current directory.

Future

This file create a rendering of your model for each of the view specified with a set of given settings.

With the same syntax is possible to automate many tedious tasks and speed up sensibly the radiance simulation.

We leave to the reader the task to investigate further.