Published on **The O'Reilly Network** (http://www.oreillynet.com/)
http://www.oreillynet.com/pub/a/mac/2001/12/14/terminal_one.html
See this if you're having trouble printing code examples

Learning the Mac OS X Terminal: Part 1

by Chris Stone
12/14/2001

Editor's note -- After reading the chapters Chris Stone contributed to Mac OS X: The Missing Manual, I asked him to write a couple of articles for the Mac DevCenter because I believe that understanding the Terminal application adds value to Mac OS X. These tutorials give you a preview of what Chris has covered in the book.

Mac OS X's Terminal application. There it sits in your Utilities folder, foreign and mysterious. You've heard that it's a portal to the new world of the Unix command line, a world where your flurries of mouse clicks can be replaced with a just few keystrokes.

But you've been wary of rushing into this new territory where the keyboard is king, concerned that without enough knowledge you might get lost, or stuck, or worse. Or maybe you're an adventurer who is just waiting to dive into uncharted waters.

This article is for you. Regardless of why you've previously avoided `[localhost:~] yourname%`, I'll show you how to take your first steps with the Terminal application. Then, I'll walk you through a tutorial that will accelerate your understanding of the Unix command line.

In Part 1 of this series, you'll learn more about what Terminal does and get an overview of the tutorial procedure. You'll then jump into the tutorial itself to learn the fundamental Unix commands you'll need to know to get started with just about any command-line procedure.

Then, in Part 2, you'll finish the rest of the tutorial, as well as learn a few more things you can do with the command line.

The command-line interface

The command-line interface (CLI) displayed in Terminal's windows provides access to the Unix shell, which is really just another way to interact with your Mac. The other method that you're probably more comfortable with is the Aqua interface. Aqua enables you to click on icons and menus, and to launch graphical applications by telling the Mac what to do.

The shell, on the other hand, allows you to type text commands to accomplish much of the same work. Typically, these typed commands launch tiny, single-duty Unix applications that do specific jobs and then quit. The shell itself is an application that plays the go-between for the commands that you enter and the Unix kernel at the core of Mac OS X. There are in fact several shells available. By default Mac OS X uses a shell called `tcsh`.

If you're curious about why you would want to use the shell in the first place, see the article Why Use a Command Line Instead of Windows? for more information about the CLI vs. the Aqua interface.

The procedure

To help you learn the Terminal application more quickly, I'm going to introduce you to a Unix utility built right into your Mac OS X system. Working with this utility will help you get more comfortable with the core Unix commands.

Installed with Mac OS X is a mechanism that performs important fine-tuning of your system. It's called `cron`. By using this Unix task-scheduling utility, you can have your system regularly purge itself of outdated, space-hogging log files, update system databases so utilities like `locate` can work effectively, and do several other maintenance tasks that keep your system running lean and mean.

The `cron` utility fully automates this process, meaning that once everything is configured, the housecleaning will happen unattended as scheduled. The good news is that Apple has done the configuration for you. The not-so-good news is that they've scheduled these groups of tasks, or `cron` jobs, to run between 4:00 and 5:00 in the morning -- a time when your Mac is likely not even on! And if your Mac is never on during these times, these important tasks will never happen.

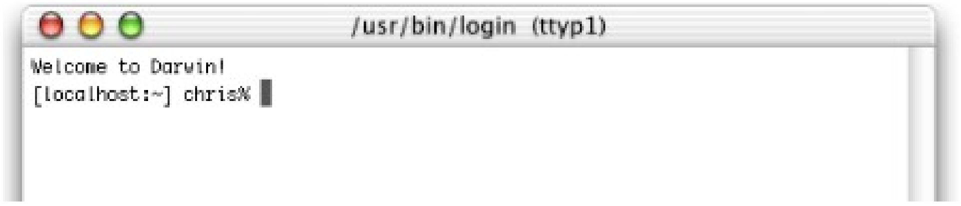
In this tutorial, I'll show you first how to modify the `cron` schedule, which is read from a file called the `crontab`, so that these tasks occur at more reasonable times. I'll then explain how to configure Mac OS X's built-in mail server and the Mail application so that you'll receive an emailed report every time the `cron` jobs run!

The tutorial

For this tutorial, make sure you're running Mac OS X 10.1 or newer, and that you're logged in with an administrator's account, though not the root account.

Open the Terminal program, which you'll find in the Applications_Utilities folder. Once launched, Terminal opens a single window displaying a greeting and a second line of text that comprises the prompt. With that window active, anything you type will enter just before the rectangular cursor that follows the prompt. After you type a command, simply press Return or Enter to run it.

The prompt shows the name of your computer (or rather its host name, which can vary), and then identifies your current working directory ("directory" is just the Unix term for "folder.") The current working directory is "where you are," that is, the location in your filesystem hierarchy that your next command will act on. Your initial working directory is always your "home" directory, which is identified in the prompt by the home directory shortcut character "~".



```
Welcome to Darwin!
[localhost:~] chris%
```

To fully display the path to your working directory, use the `pwd` command: Type `pwd` (which means "print working directory," though it only displays it) and press Return:

```
[localhost:~] chris% pwd
/Users/chris
[localhost:~] chris%
```

As you can see, `pwd` does its job by displaying the full path, or path name, to your home directory and providing you with a new prompt when done. This path name begins with the slash character, which represents the root or top-most directory of your filesystem. Note that directories that reside on your system disk do not include that disk's name in their pathnames.

For those of you trying the command line for the first time, how's it going? As for you experienced Unix users, how does the Mac OS X experience compare?

Post your comments

To act on a different set of files, you simply change your working directory using the `cd` command. We'll first be modifying the `crontab` file, which exists in the `/etc` directory (normally invisible to the Finder). Enter `cd` followed by a space and the path name of the target directory, `/private/etc`:

```
[localhost:/etc] chris% cd /private/etc
[localhost:/private/etc] chris% ls
```

Notice the change in the prompt reflecting the new working directory. If you're curious about what your working directory contains, use the `ls`, or list command:

```
[localhost:/private/etc]  chris%    ls
afpovertcp.cfg             hosts     rc.common
appletalk.cfg              hosts.equiv   resolv.conf
appletalk.nvram.en0        hosts.lpd   rmtab
appletalk.nvram.en2        httpd     rpc
authorization              iftab     services
bootstrap.conf             inetd.conf   shells
crontab                    inetd.conf~   slp.regfile
```

As you can see, there are a lot of items -- quite a bit more than what's shown here -- in `/private/etc`, including `crontab`.

The crontab file

The `cron` application launches automatically at system startup and runs continuously in the background executing commands as instructed by the `crontab` files. These files tell `cron` exactly what commands to run and when to run them. In fact, each user account can have its own `crontab` file. The system `crontab` found in `/private/etc` belongs to the super-user, or root account, and therefore can specify commands requiring the same total system access allowed to root.

cp

Before you modify the system `crontab`, you should first make a backup copy in case you need to revert back to its default state. You'll use the `cp`, or copy command, to do this, which lets you copy and rename a file in one step. Normally, to rename and copy a file into the same directory, you would type `cp`, followed by the name of the original file, and then the name of the copy:

```
[crontab: /private/etc] chris% cp crontab crontab.bak
cp: crontab.bak: Permission denied
```

But hold on. It looks like you don't have permission to write to the `etc` directory. In fact, only root can write to `/private/etc`. So, because you are not logged in as root, it might seem that there's no easy way to write to this directory. But there is....

sudo

The `sudo` utility, for "substitute-user do," allows you to gain temporary root privileges on a per-command basis. To use `sudo`, simply preface the command you wish to run as root with `sudo` and a space, and `sudo` will prompt you for your password (not root's). If you have administrator privileges, entering your password will run the `sudo`'ed command as if root were doing it.

Warning: Use `sudo` with care. You can easily make mistakes with `sudo` that could require complete re-installation of the OS to get going again. If that thought makes you queasy, it would be wise for now to use `sudo` only as directed in this article.

To perform the previous command successfully, preface it with `sudo`:

```
[office_g4:/private/etc] chris% sudo cp crontab crontab.bak
Password:
[office_g4:/private/etc] chris%
```

Notes about `sudo`:

- The first time you run `sudo`, you'll see another reminder to use `sudo` with care.
- You'll only need to enter your password when you haven't already used `sudo` within the last 5 minutes.
- It's not necessary to activate the root account to use `sudo`.

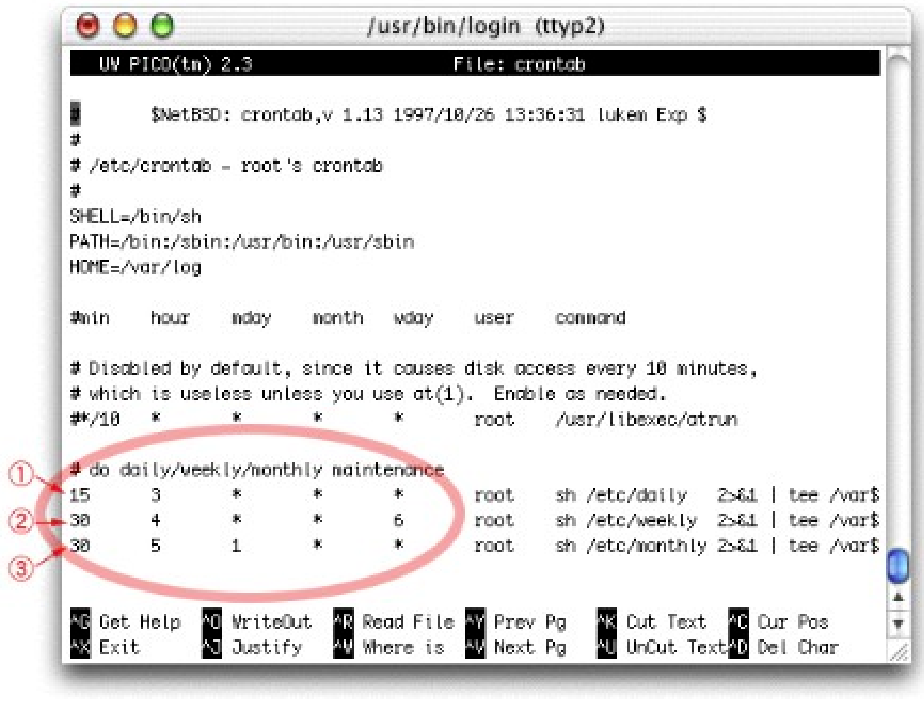
What you need to do next, then, is edit this system `crontab` file, and you'll learn how to do it with a command-line text editor called `pico`. However, if you were to first examine the privileges for `/etc/crontab`, you would see that it's owned by root, and only root has write privileges. Sounds like another job for `sudo`!

pico

Of the several CLI text editors included with Mac OS X, `pico` is the easiest to learn. To open a text file in `pico`, simply enter the file name after the `pico` command. Used with `sudo` then, the command to edit the `crontab` file in the `/etc` directory looks like this:

```
[localhost:/private/etc] chris% sudo pico crontab
```

And this is what you'll see when you run it:



```
  UW PICO(tm) 2.3                    File: crontab

#       $NetBSD: crontab,v 1.13 1997/10/26 13:36:31 lukem Exp $
#
# /etc/crontab - root's crontab
#
SHELL=/bin/sh
PATH=/bin:/sbin:/usr/bin:/usr/sbin
HOME=/var/log

#min    hour    mday    month   wday    user    command

# Disabled by default, since it causes disk access every 10 minutes,
# which is useless unless you use at(1).  Enable as needed.
#*/10   *       *       *       *       root    /usr/libexec/atrun

# do daily/weekly/monthly maintenance
15      3       *       *       *       root    sh /etc/daily   2>&1 | tee /var$
30      4       *       *       6       root    sh /etc/weekly  2>&1 | tee /var$
30      5       1       *       *       root    sh /etc/monthly 2>&1 | tee /var$


^G Get Help  ^O WriteOut  ^R Read File ^Y Prev Pg   ^K Cut Text  ^C Cur Pos
^X Exit      ^J Justify   ^W Where is  ^V Next Pg   ^U UnCut Text^D Del Char
```

The document's text area lies between the black title bar at the top and the two rows of command prompts at the bottom. The Terminal window's scrollbar won't let you scroll through the document. Instead, you use the down-arrow to move the cursor down line by line, or use the Page commands.

All of the commands listed at the bottom are prefaced with the caret character ("^"), representing the control key. So for example, to go to the next "page" (actually screen-full) of text, press the control and "V" keys as indicated. For brief descriptions of all the commands, read the `pico` help file by pressing control-G.

The numbers in the circled area specify the time `cron` runs the scripts (there are actually three of them), and this is where you'll make your changes.

Each of the three lines (numbered 1, 2, and 3) specifies one of the three scripts `cron` runs by default. Each script is different, performing its own appropriate set of maintenance procedures. The "daily" script, specified on the line labeled 1, runs once each day. The "weekly" script, specified on line 2, runs once each week. And the monthly script, specified on line 3, runs -- you guessed it -- once each month.

The first five columns or "fields" of each line specify at exactly what interval the script will run. The fields specify from left to right: minute, hour (on a 24-hour clock), day of the month, month, and weekday (numerically, with Sunday as 7). Asterisks used instead of numbers in these fields mean "every."

For example, line 1 specifies a time of 3:15 a.m.:

```
15 3 * * * root sh /etc/daily   2>&1 | tee /var$ …
```

Since the rest of the columns contain asterisks, the daily script (which is written in a file named on that line by its path name `/etc/daily`) will run at "3:15 a.m. on every day of the month, on every month, and every day of the week," that is "every day at 3:15 a.m."

Line 2 specifies that the weekly script runs at 4:30 a.m. on every weekday number 6, or Saturday:

```
30 4 * * 6 root sh /etc/weekly  2>&1 | tee /var$ …
```

And line 3 specifies that the monthly script runs at 5:30 a.m. on day 1 (the first) of each month.

```
30 5 1 * * root sh /etc/monthly 2>&1 | tee /var$ …
```

By just changing these numbers, then, you can have these scripts run at more reasonable times. Of course, what's "reasonable" depends on your own situation, so consider these factors when deciding:

1. Choose a time when your Mac is likely to be on (and not asleep).
2. Choose a time when a few minutes of background activity won't disturb your work too much. On faster machines especially, the activity is hardly noticeable, but it could cause some stuttering if, for example, you happened to be watching a DVD at the time.
3. Choose a time that is unique for each script. You don't want to schedule scripts to run at the same time.

For example, these times might be good for a machine that's only on during normal work hours:

- Daily -- every day at 5:15 p.m.
- Weekly -- every Monday at 8:50 a.m.
- Monthly -- the first of every month at 9:30 a.m.*

*(Of course, the first of the month sometimes falls on a weekend or holiday, but for now, that's the best you can do. You'll find a work around to this problem in Part 2 of the article.)

To modify the `crontab` file to reflect these new times, use the cursor keys (the four arrow keys) to move the cursor to the proper field. Except for being unable to use the mouse, you'll find that editing text with `pico` is similar to doing so with any GUI text editor. Use the delete key as usual, and type in the new values.

First, change the 3 in the daily script line to 17:

```
15 17 * * * root sh /etc/daily   2>&1 | tee /var$ …
```

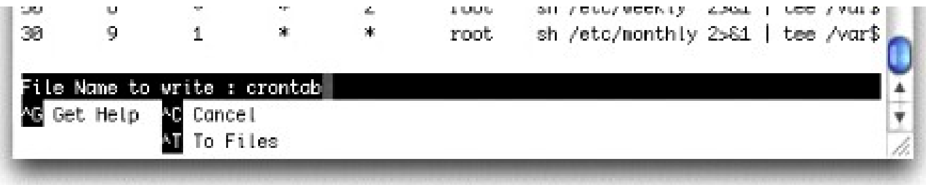
Next, change the time in the weekly script line as shown, and the day from 6 to 2 (Saturday to Tuesday).

```
50 8 * * 2 root sh /etc/weekly  2>&1 | tee /var$ …
```

Finally, change the time in the monthly script line as shown:

```
30 9 1 * * root sh /etc/monthly 2>&1 | tee /var$ …
```

Once you've made the changes, save ("write out") the document by pressing control-O. You'll then be prompted to confirm the save. Just press Return to do so.



```
30      9       1       *       *       root    sh /etc/monthly 2>&1 | tee /var$

File Name to write : crontab
^G Get Help   ^C Cancel
              ^T To Files
```

Finally, quit `pico`, by pressing control-X.

Once you've saved the `crontab` file, the new scheduling takes effect; there's no need to restart. For now, you'll not receive notification of the completed `cron` jobs, but in Part 2, you'll learn how to make that happen, as well as learn more about the scripts themselves.

Chris Stone is a Senior Macintosh Systems Administrator for O'Reilly and contributing author to Mac OS X: The Missing Manual, which provides over 40 pages about the Mac OS X Terminal.

Return to the Mac DevCenter.

Published on The O'Reilly Network (http://www.oreillynet.com/)
http://www.oreillynet.com/pub/a/mac/2002/01/22/terminal_pt2.html
See this if you're having trouble printing code examples

Learning the Mac OS X Terminal, Part 2

by Chris Stone
01/22/2002

In Part 1 of this series, you learned how to reschedule default system `cron` jobs by modifying the system `crontab`. Here in Part 2, I'll show you how to configure `cron` to email you a report each time it runs one of these jobs.

First, let's take another look at the pertinent lines in `/etc/crontab`. Since you only want to look at the file and not modify it, you don't need to use a text editor like `pico`. Instead, to only display short files like this, use the `cat` utility, which dumps the entire text file into your window and exits:

```
[localhost:~] chris% cat /etc/crontab
```

You might notice a couple of interesting things about this command line. First, since the permissions for `/etc/crontab` allow anyone to read it (but only root to modify it), using `sudo` is not necessary.

Second, this command line allows you to access a file in a directory other than your working directory by specifying, in this case, the full (or *absolute*) pathname to that file. (You didn't first `cd` to `/etc` to open `crontab`, as you did in Part 1.) An absolute path describes the directory hierarchy from the very top of the filesystem (the root directory or "/"), down to the file.

Finally, because some of the `crontab` lines might be too long for your window, you'll need to widen the window by dragging its lower right corner to the right until all lines fit without wrapping.

Look now at the line specifying the daily `cron` job, which, depending on how you configured the time segment in Part 1, begins something like this:

```
15    17    *    *    *    root    sh /etc/daily ...
```

Following the first five time fields is the user field. In this case the user field tells `cron` to run the job as if "root" were doing it. This is necessary here because these maintenance procedures require the full access to the system allowed to the root account.

Following the user field is the sixth and final field, which specifies the actual command for `cron` to execute. The heart of the command, `sh /etc/daily`, tells `cron` to use `sh`, or the Bourne shell, to run the shell script called "daily," found in the `/etc` directory.

A shell script is a file holding a series of command lines that can be batch-executed by the shell. Much like an AppleScript, a shell script allows you to create and save long automated procedures that you can run at any time, using a single command. The `/etc/daily` shell script was written for the Bourne shell, which uses command syntax more apt for shell scripting than the `tcsh` shell you've been using.

The entire daily `cron` command line, then, looks like this:

```
sh /etc/daily   2>&1 | tee /var/log/daily.out
   | mail -s "`hostname` daily output"  root
```

`cron`, in fact, uses the Bourne shell by default to execute its `crontab` commands, so this command wouldn't work if you ran it manually using the `tcsh` shell. For the purpose of this article, though, you won't need to modify this command. You won't need to fully understand it either, but this breakdown will give you a general idea of what it does:

```
sh /etc/daily  2>&1 |
```

Run the `/etc/daily` shell script using the Bourne shell, and send its output and any of its error messages on to the next command.

```
tee /var/log/daily.out |
```

Write the input from the previous command to a file and also pass it on to the next command. (That's right -- you can always see the results of the latest daily `cron` job by viewing `/var/log/daily.out`.)

```
mail -s "`hostname` daily output"   root
```

This is the place for your questions and comments as you work with Mac OS X's Terminal application.

Post your comments

Using the input from the previous command as the body, create an email message with the subject "localhost daily out" and send it to root. This, then, is the command that starts the `cron` report's journey on its way to your mailbox.

The first step in getting the reports delivered is to direct them to *your* mailbox, instead of root's. To do this, you could conceivably replace "root" in this command with your account name, but wouldn't it would be simpler if you could just tell the system to redirect *all* mail addressed to root to your mailbox instead? By using a `.forward` file, you can do just that.

A `.forward` file is just a text file containing nothing but the name of the account to which you want the mail forwarded (in this case, your account name). Once the `.forward` file is placed in the home directory of the original addressee (in this case, root), the mail gets forwarded as expected.

By default, the Mac OS X root home directory already contains a `.forward` file, but this one redirects mail not to another user, but into thin air. This happens because instead of an account name, root's `.forward` file contains the pathname `/dev/null`, which is the location of a Unix black hole. Streams of data directed to `/dev/null`, mail messages included, simply disappear. Since OS X's designers figure most users won't be accessing root's mail, they used this method to ensure mail doesn't pile up at the door while no one's home, eventually filling up your hard disk.

Let's get back to Terminal, then, to edit root's `.forward` file. You've probably already noticed that there is no "root" directory in `/Users` -- so where is root's home? The easiest way to find any user's home directory is by using the `~` shortcut (for home directory), along with the account name. Therefore, to change your working directory to root's home directory, enter this:

```
[localhost:~] chris% cd ~root
[localhost:~root] chris%
```

Okay, so now you're in `~root`, but where is that really? Remember, you can find out with `pwd`:

```
[localhost:~root] chris% pwd
/private/var/root
```

Next, use `ls` to view the contents of `~root`:

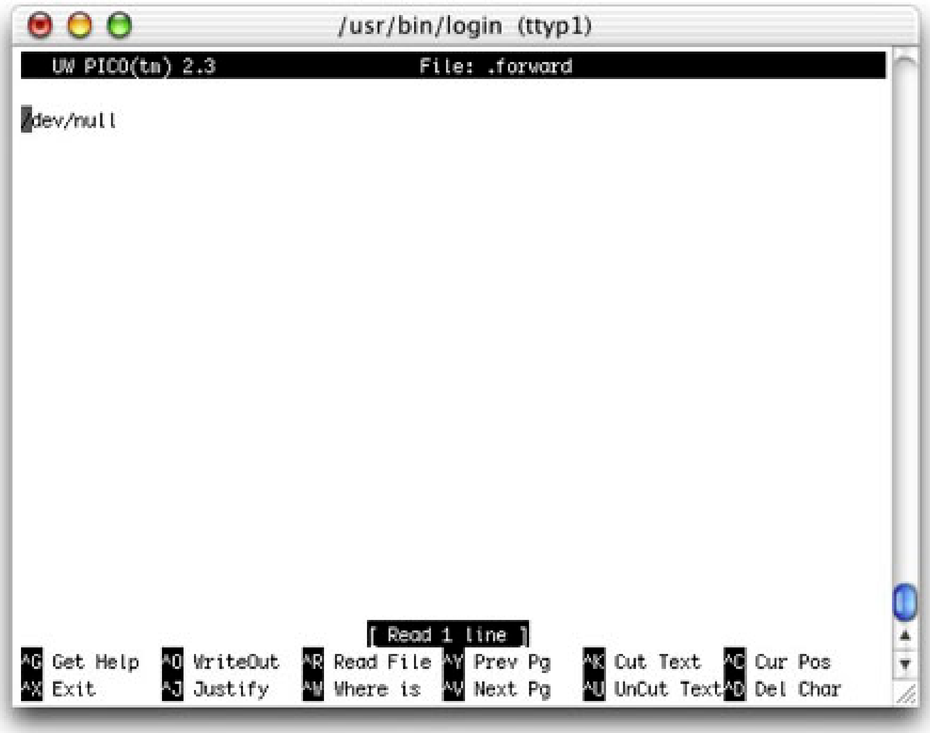
```
[localhost:~root] chris% ls
Desktop   Documents Library
```

The contents of *your* `~root` directory might look different, but in any case, you still won't see a `.forward` file. What happened to it? One clue is the first character of the file's name, which is a dot ("."). An initial dot is the Unix way of marking filesnames as invisible to the shell (and to the Finder as well). But it's easy enough to see such hidden files in the CLI; just use the `-a` option flag with `ls`.

Option flags allow you to modify the behavior of commands. The `-a` option flag for `ls` tells `ls` to display "all" items in the working directory, including hidden ones (also known as "dot files"). Simply type the `ls` command, add a space, and then type the `-a` flag:

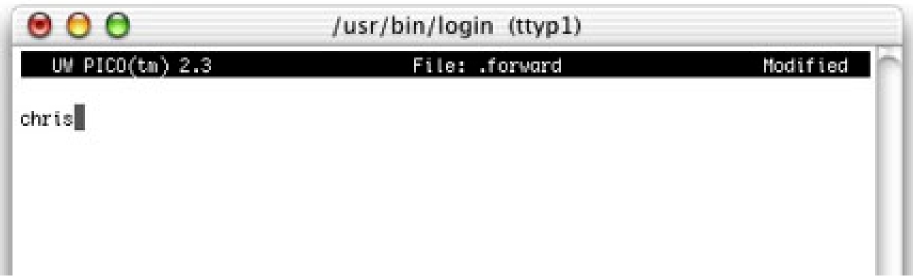
```
[localhost:~root] chris% ls -a
.              .forward  .tcsh_history    Library
..              .nsmbrc   Desktop
.CFUserTextEncoding   .ssh   Documents
[localhost:~root] chris%
```

Again, your view might be different, but you will now see `~root/.forward`. You'll next want to edit it using `pico`. Since this file is owned by root, you'll need to use `sudo` as well:


```
/usr/bin/login (ttyp1)
  UW PICO(tm) 2.3                    File: .forward

/dev/null




                                 [ Read 1 line ]
^G Get Help  ^O WriteOut  ^R Read File ^Y Prev Pg   ^K Cut Text  ^C Cur Pos
^X Exit      ^J Justify   ^W Where is  ^V Next Pg   ^U UnCut Text^D Del Char
```

```
[localhost:~root] chris% sudo pico .forward
```

First, delete the file's single line by pressing control + K. Next, type in your account name (the name that's just before the "%" in the prompt; "chris" in this case):


```
/usr/bin/login (ttyp1)
  UW PICO(tm) 2.3                    File: .forward                 Modified

chris
```

As you learned in Part 1, save the file with control + O, press return to confirm the name, and then press control + X to exit `pico`. You're done with the `.forward` file.

The next step in the procedure involves getting the mail transfer agent `sendmail` to launch successfully when beckoned by the mail user agent `mail` (which, as you remember, is invoked by the `cron` job).

Mail user agents (MUAs) are the kinds of applications that allow you to personally send, receive, and otherwise work with your email messages. Outlook, Eudora, and Mac OS X's Mail application are other familiar examples of MUAs.

Mail transfer agents (MTAs), on the other hand, are the not-so-familiar applications that receive the messages from the MUA and pass them on to other users on the same machine, or to MTAs on other machines for ultimate delivery to users elsewhere. The `sendmail` MTA included with OS X is one of the most popular, used on servers large and small across the Internet.

Since you are not running your own mail server at this point, you don't need `sendmail` to be running at all times. Instead, you only need to ensure that `sendmail` launches when invoked by `mail` to send the `cron` reports. Used this way, `sendmail` quits itself once delivery is made.

For `sendmail` to successfully launch, however, one issue needs to be "fixed" on your system. As a security measure, `sendmail` will not run with OS X's default permissions (termed "privileges" in the Finder), namely those for the root directory.

This fix involves one simple change: eliminating write privileges for the group assigned to the `root` directory. The CLI makes it very easy to view and change the various permission settings for any item, but the procedures are still too involved to detail here. (Of course, Mac OS X: The Missing Manual does include an in-depth look at permissions and the CLI.)

Instead, I'll zero in on the single command line required to get `sendmail` going:

```
sudo chmod  g-w /
```

Since you're modifying the settings for a root-owned directory, the command line starts with `sudo`.
Next comes the `chmod` command, for "change (file) modes." File modes are the settings that specify whether an item can be read or written to, for example, and by which kind of user -- the owner of the item, its group, or any user of the machine. (These settings correspond, of course, with the Privileges settings that are accessible via Finder's Inspector.)

Following a space are `chmod`'s "arguments," the first of which specifies the modes to be changed (option flags are just another kind of argument, by the way). This argument says to take the group ("`g`") and remove ("`-`") its permission to write ("`w`") to the file or directory specified in the next argument (again followed by a space), which in this case is the root directory, ("`/`").

Your next step, then, is to run the command line:

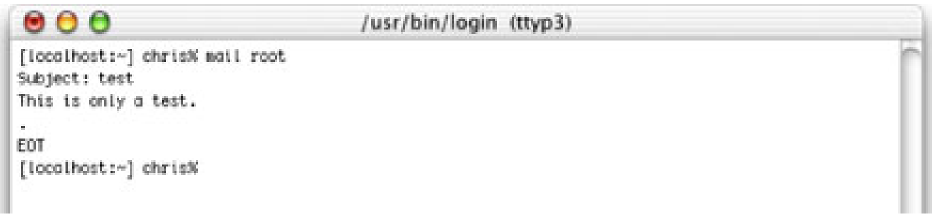
```
[localhost:~] chris% sudo chmod  g-w /
```

Once you do, `sendmail` should work fine. However, you should know that Mac OS X upgrade installers and some application installers change the root directory back to group-writable, so you'll need to run the `chmod` command line whenever this happens.

To test everything so far, try sending mail from the CLI. Use the `mail` command to send mail to root (which, at this point, will get forwarded on to you) like this:

```
[localhost:~] chris% mail root
Subject:
```

You're now working inside the `mail` CLI application, so you'll see no more `tcsh` shell prompts until you exit `mail`. Enter any subject you would like and press return. Type in your message at the cursor. To end your message, send it, and exit `mail`, press return, type a period, and press return again. You'll then return to your shell prompt:


```
[localhost:~] chris% mail root
Subject: test
This is only a test.
.
EOT
[localhost:~] chris%
```

After a few moments, check your mail by entering the `mail` command again, but this time with no arguments. Until the message arrives, you'll only see that your box is empty when you run `mail`:

```
[localhost:~] chris% mail
No mail for chris
```

However, once it arrives you'll see something like this:

```
[[localhost:~] chris% mail
Mail version 8.1 6/6/93.  Type ? for help.
"/var/mail/chris": 1 message 1 new
>N  1 chris                  Sat Jan  5 15:30  13/374   "Test"
&
```

You're back in the `mail` application, but this time to view your new message. Press return at the "`&`" prompt to have a look:

```
Message 1:
From chris  Sat Jan  5 15:30:01 2002
Date: Sat, 5 Jan 2002 15:30:00 -0800 (PST)
From: Chris Stone
To: root
Subject: Test

This is only a test.

& q
Saved 1 message in mbox
[localhost:~] chris%
```

As you can see, the test message stays in your local Unix mailbox when you quit mail. Note that this and any other messages there will disappear as a result of the following procedure. However, if this tutorial is new to you, it's very unlikely that you have other messages there anyway. (Of course, your POP and IMAP mail will stay safe and sound.)

You're now ready to set up your GUI Mail application so it can access your local Unix mailbox. Since you will be modifying the folder in which Mail stores its mail, `~/Library/Mail`, you should first make a backup of it to, say, your Documents folder:

```
[localhost:~] chris% cp -R Library/Mail/ Documents/Mail
```

Here are some important points about this command line:

- Because you are copying a directory, `cp` requires you to use its `-R` option flag (for "recursive").
- The pathnames are not absolute, but "relative" to your working directory. Instead of including the entire

pathname from the root directory down, with relative pathnames you can specify a shorter path that begins from the end of your working directory. (Always remember to omit a leading `/` in relative pathnames.)

- The target pathname, `Documents/Mail`, doesn't specify the directory in which you would like `Library/Mail/` to go, but the desired new relative pathname of the copied directory.

- If this command line could talk, then, it would be telling the shell, "Please make a copy, including all contents, of the directory indicated by the first pathname. When you're done, the pathname of the new directory is to be the same as this line's second pathname."

Something else you should know about `cp` is that it does not properly copy files with resource forks, so you should never use it for that. You'll never have a problem copying Unix and Cocoa applications and related files, which don't contain resource forks, but if you are unsure, use the Finder to copy (or have a look at Mac OS X: The Missing Manual for an explanation of using CpMac, which does handle resource forks reliably).

The next step is to make the directory that Mail requires before it can create a Unix mail account. The directory must exist in `~/Library/Mail/` and be named "`UNIX:@`". To create a directory from the command line, use the `mkdir` command, followed by a space and the name of the new directory.

However, if there is no `Mail` directory already inside `~/Library`, the command will return an error. To prevent this possibility, use `mkdir`'s `-p` option flag, which will create any intermediate directories for you if they are missing.

```
[localhost:~] chris% mkdir  -p Library/Mail/UNIX:@
```

Next you'll need to open Mail and create a Unix mail account, which requires just a few simple steps:

1. Open the Mail application, found in `/Applications`.

   Sure, you can just double-click its icon to open it in the Finder, but since you're in Terminal anyway, how about opening it from there? To do so, just use the `open` command (don't forget to include the normally-hidden extension at the end):

   ```
   [localhost:~] chris% open /Applications/Mail.app
   ```

   Mail launches immediately, just as if you had opened it from the Finder.

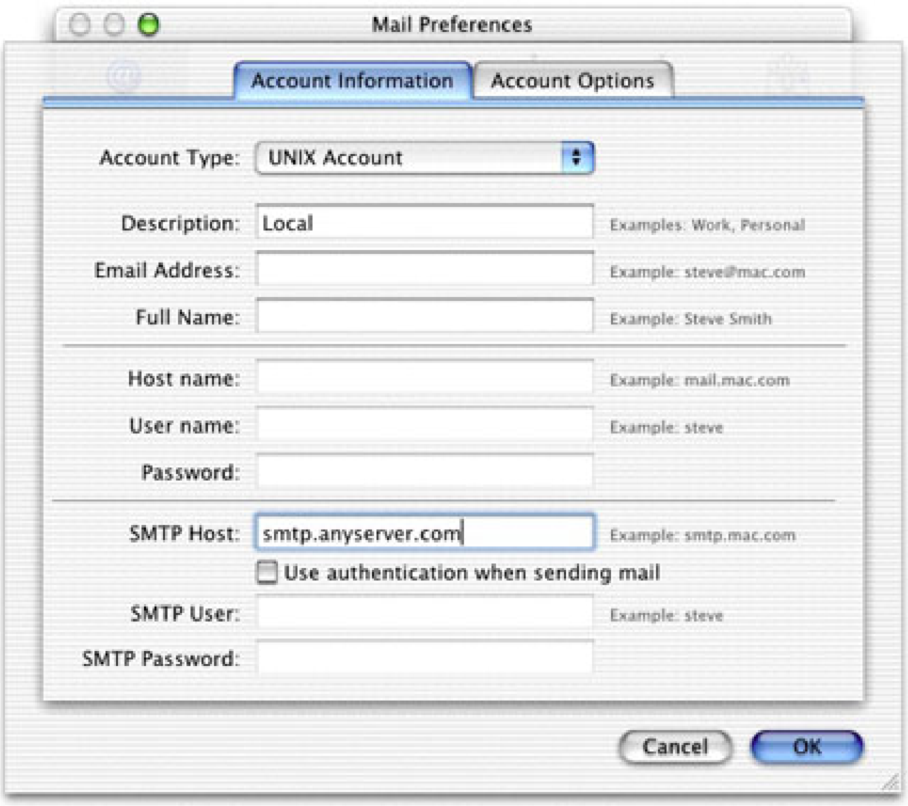
   If you've never used Mail before and have no email account info entered in your System Preferences, you'll be prompted to set up an initial account. At a minimum, you'll need to enter an email address, so enter anything you would like; it won't affect the setup of your Unix mail account.

   You can safely click through the other prompts for server and other info, and to import mail from other applications. None of this is needed for the task at hand.

2. Create a new Unix mail account.

   From Mail's Mail menu, select Preferences, and then click the Accounts icon. In the Accounts pane, click Create Account. To configure the account, you'll at least need to select the account type (Unix Account), enter a description (Local), and enter something -- anything, really -- in the SMTP Host field.

Of course, if you need to set up a bona fide Unix account, all of these fields mean a great deal. However, for the purpose of only accessing your local Unix mail, this is all you need to configure:



Account Type:	UNIX Account	
Description:	Local	Examples: Work, Personal
Email Address:		Example: steve@mac.com
Full Name:		Example: Steve Smith
Host name:		Example: mail.mac.com
User name:		Example: steve
Password:		
SMTP Host:	smtp.anyserver.com	Example: smtp.mac.com
	☐ Use authentication when sending mail	
SMTP User:		Example: steve
SMTP Password:		

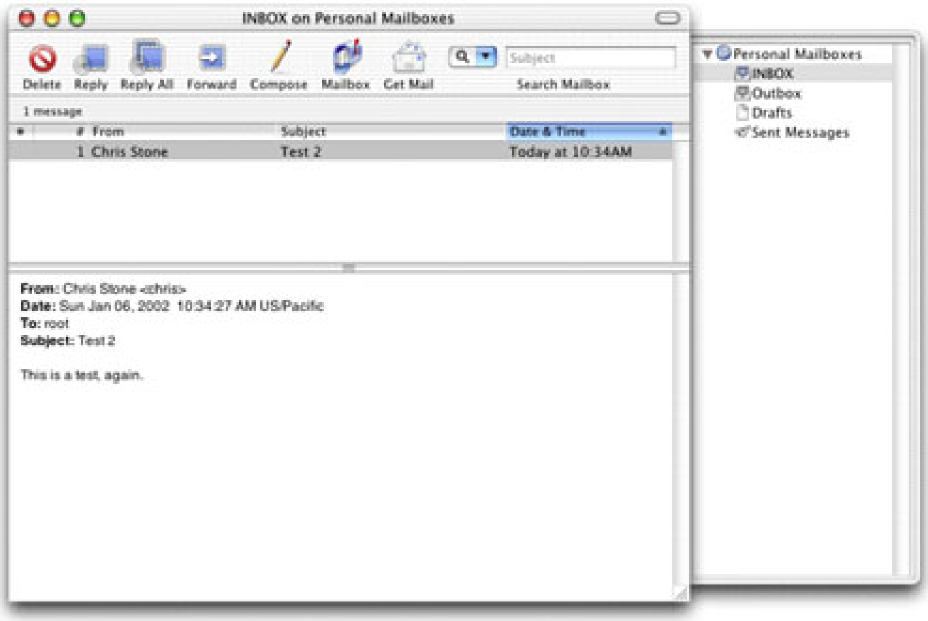
3. Click OK, close the Preferences window, and you're all set.

   If you are already using Mail to check your regular POP and IMAP accounts, this additional account will not affect those in any way, except that new mail from your Unix account will show up in your default inbox. Of course, if you would like, you could create a new mailbox and a rule to have the incoming `cron` reports be placed there instead.

Now that everything's in place, you can perform a test. Send a new mail message to root:

```
[localhost:~] chris% mail root
Subject: Test 2
This is only a test, again.
.
EOT
[localhost:~] chris%
```

Switch to Mail, and then click Get Mail until you see the message has arrived in your Inbox:



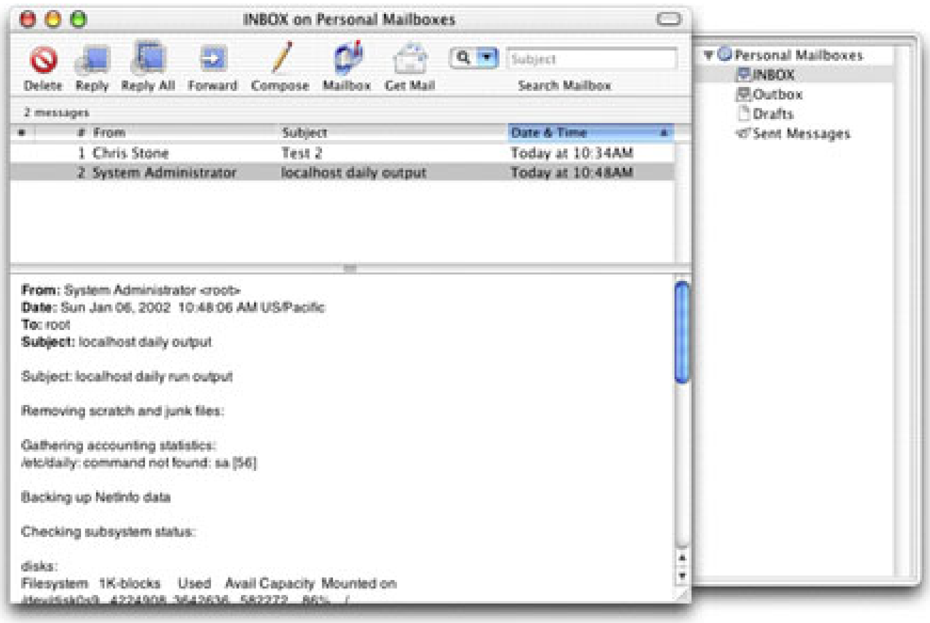
1 message

#	From	Subject	Date & Time
1	Chris Stone	Test 2	Today at 10:34AM

From: Chris Stone <chris>
Date: Sun Jan 06, 2002 10:34:27 AM US/Pacific
To: root
Subject: Test 2

This is a test, again.

If you see the test message in your inbox, then you're done. The next time `cron` runs one of the maintenance jobs, you'll see the report in your inbox as well. For example, the daily `cron` job report will look something like this:



2 messages

#	From	Subject	Date & Time
1	Chris Stone	Test 2	Today at 10:34AM
2	System Administrator	localhost daily output	Today at 10:48AM

From: System Administrator <root>
Date: Sun Jan 06, 2002 10:48:06 AM US/Pacific
To: root
Subject: localhost daily output

```
Subject: localhost daily run output

Removing scratch and junk files:

Gathering accounting statistics:
/etc/daily: command not found: sa [56]

Backing up NetInfo data

Checking subsystem status:

disks:
Filesystem   1K-blocks    Used   Avail Capacity  Mounted on
/dev/disk0s9   4224908 3642636  582272    86%    /
```

Now that these regular reports will be coming in, you'll probably want to be able to understand them. In Part 3, you'll get a closer look at the scripts themselves to learn how to read the reports they generate.

Also in Part 3, I'll show how a Macintosh with a persistent Internet connection can send its reports to any email address. Until then, keep checking to see that you're receiving the reports as expected, and always feel free to submit your comments or questions to our TalkBack section.

I'd like to thank Scott Gever for his techical help with this series.

Chris Stone is a Senior Macintosh Systems Administrator for O'Reilly and contributing author to Mac OS X: The Missing Manual, which provides over 40 pages about the Mac OS X Terminal.

Return to the Mac DevCenter.