An Inessential Guide to Athena
Fifth Edition

The Student Information Processing Board

February 12, 2001

Contents

Purpose and Conventions

We of the Student Information Processing Board (SIPB) have compiled this document to provide a catalog of the less-publicized commands and capabilities of the Athena system for people who have learned the basics of Unix. If you are a complete newcomer to the system, we suggest that you begin with the Athena documents. Some information contained in earlier versions of this *Guide* is now covered. These documents are available at Graphic Arts in the basement of building 11 or the Student Center.

File names and "meta-names" throughout this document are indicated in *italics*.

Program names, commands to be typed to the shell, and text that is returned by a command are in `typewriter text`.

Document names are in `typewriter text`.

Commands for GNU Emacs and other interactive programs are in **boldface**.

A C- prefix indicates Control character, respectively. For instance, C-x means that you hold down the Control key and hit the "x" key. A M- prefix indicates a Meta character, so for M-a you would hold down the "Alt" (or "Compose Character" on some keyboards) key and type "a".

If you have any questions, please feel free to ask us.

SIPB and /mit/sipb: who? what? when? where? why? how?

SIPB (pronounced "Sip-bee") is the Student Information Processing Board, a student group concerned with computing at MIT. We administer several machines and an AFS cell, provide Usenet access to Athena, are available for telephone (253-7788) or in-person (W20–557) consulting at almost any time of day or night, have one-of-a-kind meetings Monday evenings at 7:30 pm, write documentation (like this guide), hack, and generally have a good time. We also act as an advocate for student computer users and student computer access on campus.

SIPB maintains a locker, `/mit/sipb`. To access the programs and files on it, you type `add sipb`. This locker is the place where SIPB-written or modified programs and documentation can be found. The source code for most of the programs in `/mit/sipb` can be found in the `/mit/sipb/src` subdirectory. We hope that people will find the programs useful and the source code instructive.

Since Athena provides SUN SparcStation, SUN UltraSparc, SGI Indy, and SGI O2 workstations, as well as X86 machines running Linux and NetBSD, for public use, most SIPB programs are compiled for use on all these platforms. In addition, some SIPB programs are compiled for the Sun4 machines running SunOS 4.x, NeXT machines, DEC DECStation, IBM RS/6000, and HP 9000/700 series computers, as well as the older DEC VAXstation and IBM PC/RT computers.

Another locker which SIPB offers is `/mit/outland`. It is *not* supported, and while it contains many nifty programs, you should not depend on any of them, as they may not necessarily be available or completely debugged. We encourage you to look at it despite this, since it contains many neat programs that SIPB doesn't have the resources to officially support.

There are many other `add`able lockers with interesting and useful programs on them; it is not possible to catalog all of them here. We can only suggest you ask your friends who might know where the interesting "hacks" are stored, or ask us if you have specific questions. However, there is a short list of some useful lockers presented later in this document.

Problems or questions with *sipb* programs should be sent to `bug-sipb@mit.edu`, except for problems with Usenet news programs, which should be sent to `usenet@mit.edu`.

MH Alternatives: *mh-rmail*, *rmail*, and *pmail*

Currently, MIT Information Systems (IS) supports the *MH* mail handling system on Athena and Eudora for Microsoft Windows systems [1]nsert legal garbage here connected to MITNet. In addition to the standard MH and *xmh* interfaces, there are at least three other major mail reading systems available on Athena workstations: *mh-rmail*, *rmail*, and *pmail*. Both *mh-rmail* and *rmail* run directly inside of GNU Emacs, which is a distinct advantage in some ways - namely, you don't have to wait for an editor to start up if you wish to compose or edit mail. The third, *pmail*, is a version of the Berkeley Unix `mail` command that understands the KerberosTM and POP systems used for mail here at MIT. It will be primarily of interest to people who are already experienced with Berkeley `mail`.

Mh-rmail

Mh-rmail is a "front-end" for MH, the standard Project Athena mail reading system. This section assumes that you are familiar with MH, at least at the level of **Essential Mail**, as well as familiar enough with GNU Emacs to be able to use multiple buffers.

The advantage of using *mh-rmail* is that you don't have to wait for an editor to start up (like `comp`); the editor is there already! And the commands are familiar to most people who are used to MH.

Starting *mh-rmail*: *Mh-rmail* exists only inside GNU Emacs; in Emacs, type the command `M-x` *mh-rmail*. You should first see the message *Loading mh-e...done*, and then *inc +inbox...done*. You are now in an "mh folder buffer," called *+inbox*, showing the summary lines for the new messages (not coincidentally, the output from the *inc* command). The folder initially only contains the new summary lines. If you wish to see all the mail you have in the folder, type `M-r` to "rescan" the folder. If you get new mail while in a folder buffer, use the `i` command to "inc" new mail.

Commands for reading your mail There are two main states of *mh-rmail*: one-window summary mode, and two-window mail-reading mode. To switch between them, use the *t* (for "toggle") command. When you do this from one-window summary mode, the screen splits into two windows. The window on the top contains the folder buffer, and the window on the bottom contains a *mh-show* buffer, containing the text of the message. The cursor remains in the top window, on the summary line of the current message. If it gets out of agreement with the show buffer (for example, if you use `C-c` *n* and `C-c` *p* instead of *n* and *p* to move through it), just hit . (That's right, a period.) to synchronize the two windows.

You can use *SPACE* to page the message up a screenfull, *DEL* to page down a screenful.

Reading other messages You can look at other messages using the *n* and *p* commands to go forward and back one message; like most other commands, these accept a repeat count with `C-c` *u*.

[1]i

You can get a hard copy of a message by using the *l* (ell) command. You must specify the printer by changing *lpr-switches* as described above.

Deleting and refiling messages *Mh-rmail* works on a "batch" principle for alterations to the state of a folder; you do a number of commands in the folder buffer to mark messages for deletion or refiling in other folders, and then you type the *x* command to actually process the commands.

d marks a message for deletion, and ˆ (caret) marks it for refiling (you will be prompted for the name of a folder to refile it in). *!* refiles the current message in the same folder as the last refile. *u* undoes any action which was marked for the line.

To actually execute all of the commands, use the *x* command. You should do this before exiting Emacs, as marked deletions don't actually occur until you do the *x*.

Sending mail There are several ways to send mail. All initially ask a few questions, then give you a window with a partial draft to play with. The simplest is the *s* command (which is like the shell command `comp`). Another way to get at this command from outside of *mh-rmail* is with `M-x` *mh-smail*. It first prompts for the "To:", "Cc:" and "Subject:" lines for the new message, and then gives you a draft with those lines filled in. *A* (answer) is related; it asks whether you want to reply to the sender ("from"), the sender and the people on the To: list ("to"), or the sender and all recipients ("cc"); it fills out the draft appropriately and sets the subject to be "Re:" plus the old subject. *F* is used to forward mail to other parties; it prompts for "To:" and "Cc:", and sets the subject to be "[Original Sender: Original Subject]". The most bizarre (and possibly most useful) one is `M-e` (Extract rejected mail), which allows you to extract the original message from a message bounced from MAILER-DAEMON. You can type `C-c C-c` to send the draft when you're done with it.

Draft mode commands There are a large number of commands available in mh-letter-mode; most importantly, `C-c C-c` (already mentioned) sends the current draft.

`C-c t`, `C-c c`, `C-c s`, `C-c b`, and `C-c f` move the cursor to the end of the current To:, Cc:, Subject:, Bcc:, and Fcc: lines, respectively, in the draft. If a line with the appropriate header does not exist, it is inserted.

`C-c C-y` inserts a message from a folder into the draft. It is useful if you want to quote someone else's message.

`C-c y` is similar to `C-c C-y`, except that it doesn't prompt for the folder and letter to insert (it inserts the message that you are replying to).

`C-c q` deletes the draft, and returns you back to where you were.

`C-c w` lists the people receiving the the draft in another buffer (this runs the MH `whom` program to find out the recipients.

Other commands `M-p` can be used to renumber the folder so that all messages left in it start with "1" (one) and go on up from there; this may take a while if you have a large folder.

`M-l` (meta-ell) lists all your folders in a temporary window. `M-f` lets you choose a range of another folder to bring up in a new folder buffer; you can then use `C-X` *b* to switch between them.

`M-k` allows you to delete a folder. This can be dangerous.

Variables that can be set. These variables can be set as described above in the Emacs section.

mh-clean-message-header If set to *t* (true), it cleans out most of the garbage headers (like "Received:" and "Message-Id:") out of the message; this takes a little more time every time you look at a message, and so is set to *nil* (false) by default.

mh-lpr-command-format sets the format used for the *l* (ell) command, to print a message.

mh-summary-height sets the height in lines of the summary window, including the mode line. By default, it is set to 4.

mh-ins-buf-prefix sets the prefix inserted before each line of messages which are "yanked" into a draft. By default, it is set to "`>>`".

Rmail

Rmail is the standard GNU Emacs mail reader. It has, along with *mh-rmail*, the advantages of reading mail *inside* an editor—namely, not having to wait for an editor to start up if you wish to compose or edit mail. It is invoked by `M-x` *rmail* from within emacs. Its syntax is fairly simple. Its major disadvantage is that the format *rmail* uses to store mail is rather strange compared to most other formats[2]. Files can be archived in the standard Unix format, but this is not the default.

Commands in *rmail* are typically single-character commands, for quick and easy use (this can be dangerous, if you are unsure of your typing ability).

Starting rmail After invoking *rmail*, emacs will report on its status (e.g. *loading rmail, getting mail from post office...*) in the mini-buffer. Following this, you will begin in one of three places. If you got new mail, you start with the RMAIL buffer displaying the first screen of the first new message you received. If you didn't receive any new mail, but still have old mail in your *RMAIL* file, the RMAIL buffer will display the first screen of the last message still filed along with displaying the message *"(No new mail has arrived)"* in the mini-buffer. However, if you have no new mail and there is no old mail still in your *RMAIL* file, you will see the following message in your RMAIL buffer along with the no new mail indication in the mini-buffer:

```
BABYL OPTIONS:
```

[2]It is the Babyl mail format used on some other computer systems.

```
Version: 5
Labels:
Note:   This is the header of an rmail file.
Note:   If you are seeing it in rmail,
Note:   it means the file has no messages in it.
```

Reading mail Emacs stores the mail you receive while using *rmail* in an *RMAIL* file (by default this is *~/RMAIL*). When you start *rmail* emacs loads in the *RMAIL* file and changes to a special mode for *rmail*. In this mode, you can move forward and backward within a mail message by using *SPACE* and *DEL* respectively. Scrolling through mail messages is effected by using *p* for previous message and *n* for next message. It is also possible to specify explicitly which message to visit; this is accomplished by typing a number representing the message number of the message you wish to visit in the *RMAIL* file and then typing *j*. Typing *1 j* will move you immediately to the first message in the file; `>` will move you to the last message.

Deleting... the concept The current mail message (the one displayed in the RMAIL buffer) will be deleted if you type *d*; *rmail* will attempt to visit the next message. Alternately, you can type `C-d`; when this is used after deleting the message *rmail* attempts to visit the previous message. However, as long as you haven't expunged the deleted messages, the deletion isn't permanent. Typing *u* will attempt to undelete the current message; if it is not successful (i.e. the current message wasn't deleted), it will try earlier messages until it finds a deleted message to undelete. A deleted message could become the current message in several ways. `M-n` and `M-p` behave like *n* and *p* except they will even visit deleted messages. Also, *j* will jump to a message regardless of whether or not it has been deleted. The deletions will be made permanent when you expunge the *RMAIL* file using *e*; this is also automatically done when you exit *rmail* or save the *RMAIL* file.

Taking inventory It is possible to see what messages you have in your *RMAIL* file by using `C-M-h` or *h*. This will split your emacs buffer into two buffers. The summary buffer will show a single line listing for every message (even the deleted ones) in your file. This lists the message number, followed by a *D* if the message has been deleted, date, sender, and subject for each message. A similar summary can be generated for particular recipient(s) using `C-M-r`; you will be prompted in the mini-buffer for the recipients to include. It is possible to scroll through the summary buffer using normal emacs control keys (e.g. `C-p`, `C-n`, `C-v`, `M-v`, ...).

Ackphtpp!...there are multiple buffers displayed in my emacs `C-M-h` as well as several other *rmail* commands will split your emacs window/screen into two separate emacs buffers. It is possible to switch between the buffers by typing `C-x o`. It is also possible to make one of the buffers go away, thus returning you to single buffer mode in emacs by typing `C-x` *1* (one) in the buffer you want to keep. Don't Panic if you do this in the *wrong* buffer as you can always change buffers using `C-x` *b*. The *rmail* buffer is called RMAIL.

Sending mail Inside *rmail*, you can send a mail message simply by typing *m*. This will create a *mail* buffer with fields for To: and Subject: to be filled in and an appropriate text separator. Fill in these fields and add your text after the separator using normal emacs editing commands. To send the message, type `C-c C-c`. By typing *r* in *rmail* you can reply to the current message. This behaves much like *m* except the header lines are already filled in appropriately according to the message you are replying to. Similarly, typing *f* allows you to forward the current message. This starts a new message to be sent like *m* with the subject line filled in and the message you were visiting in the text section of the outgoing message below a line delimiting the start of the forwarded message. You should fill in the To: field and then whatever text you deem appropriate (including editing the forwarded message if you desire). Sending both replies and forwards is accomplished just as with *m* by using `C-c C-c`. If you stop working on an outgoing mail message to do something else (e.g. look at some of your mail), you can get back to the buffer where you are editing outgoing mail by typing *c* to the RMAIL buffer.

Sending mail without invoking *rmail* From emacs it is possible to send mail without invoking *rmail*. This is done by typing `C-x` *m*. Again mail is sent using `C-c C-c`.

Filing mail Yes, there are things to do with mail other than delete it. Typing *o* will prompt you for a file to write the current message out to. It will then append the current message to this file in Babyl format (creating the file if it doesn't already exist). You can then type *i* to view files written out in this manner; *rmail* will prompt you for the filename and load the file in *rmail* mode so that you can view them in the same way as when you originally received them and they were in your *RMAIL* file.

You can also write mail out into a somewhat more readable form, particularly Unix-format mail (that is **all** the headers and the message) by typing `C-o`. As with *o*, you will prompted for a file to write the message out to and then append the current message to that file. To read messages saved in this format, you type `C-u` *g*, and type in the name of the file containing the mail.

Saving mail Typing `C-x C-s` will save the *RMAIL* file without first attempting to expunge it. Using *s* saves the *RMAIL* file after expunging it.

Exiting *rmail* The normal way to exit *rmail* is to type *q*. This expunges deleted messages, saves the *RMAIL* file, and switches to the buffer you were in when you invoked *rmail* (or *scratch* if you started with `emacs -e rmail`).

Re-entering *rmail* To re-enter *rmail* from another EMACS buffer, type `M-x` *rmail*, and you will return to your RMAIL buffer. You will also receive any new mail that may have arrived.

More mail? If you receive new mail while in *rmail* or just want to check if you have more mail, type *g* to attempt to get new mail. If there is new mail, it will be added to the *RMAIL* file and you can proceed accordingly. If there is no new mail, the message "*(No new mail has arrived)*" will be displayed in the mini-buffer.

Aliasing in *rmail* Aliases can be used to give yourself a shorter or more appropriate name for someone you send mail to or to create a personal mailing list. Aliasing in *rmail* is accomplished by specifying the aliases in the file *~/.mailrc*. Aliases are of the form: `alias` *alias_name name1 [name2 [name3 .....]]*. For each alias you should have a line in your .mailrc file that starts with "`alias`," followed by the alias you are defining, set off by one or more spaces; the remainder of the line is a *SPACE*-separated list of actual email addresses to be bound to the alias being defined. Here is a sample *.mailrc* file:

```
alias quux root@deep-thought
alias foo beeblebrox@zaphod.caltech.edu
alias bar ford@eddie.mit.edu
alias baz arthur
alias frobble foo, bar, baz
```

Here, quux, foo, bar, and baz are aliased to the respective strings following them. Whenever any of these is used in the To:, Cc:, or Bcc: field of a letter sent from *rmail*, the appropriate string will be substituted when the mail is actually sent. When frobble is specified as the recipient of a message, the message will be sent to beeblebrox@zaphod.caltech.edu, ford@eddie.mit.edu, and arthur.[3] **Note:** If you change your ~/.mailrc file, changes won't take effect until you exit and re-start EMACS and *rmail*.

Other random things... If you ever have a desire to edit a message you receive, type *w*. `C-c C-c` will return you to *rmail*. `C-c C-]` will return you to *rmail* without making the changes you just made to the mail message.

Typing `M-s` will cause *rmail* to prompt you for a string, search for the string, and show the message in which it is found.

You may notice that *rmail* does not display every header you may have seen using other mail interfaces. To toggle between seeing all the headers and just the subset *rmail* uses, type *t*.

This description doesn't include **everything** that can be done in *rmail*. To get a list of all the editing commands usable in *rmail* mode, type `C-h` *m* or *?*.

[3] *rmail* can expand recursive aliases like this.

Zephyr: When you care enough to send the very best

What is Zephyr?

Zephyr is a set of programs that sends messages between users. When a message arrives for you, a window is created in the upper-left corner of your screen(by default), and the message is displayed there. Once you have read the message, you move the mouse cursor into the window and click on any button, and (poof!) the window goes away. Additional information about Zephyr can be found in *An Inessential Guide to Zephyr*, also published by the SIPB.

How do I send a message?

The command `zwrite` is used to send a message to another user (or group of users). Let's say that you want to ask your friend, whose username is `friend`, if she's taking 6.001 this term. You type

```
% zwrite friend
```

Zephyr will check to see if `friend` is logged in anyplace. If it can't find her, you'll get the response

```
friend: Not logged in or not subscribing to messages
```

If she is available, though, you will be able to send her your message. When the message is sent you will get the message:

```
friend: Message sent
```

How do I send to more than one person?

If you want to send the same message to George Washington and Abraham Lincoln, you could type

```
% zwrite gwashing alincoln
```

and then type the message as usual. Each would think that the message had been sent to him alone, unless you tell them in the message "I'm sending this to both of you."

But what if you want to send messages to all the presidents, and you don't want to leave out any? The answer is simple — Zephyr has what is called an **instance**.' You can send a message to everyone "subscribing" to the instance "presidents" by typing

```
% zwrite -i presidents
```

How do I "subscribe" to these "instances"?

There is a program called `zctl` that controls Zephyr. Typing `zctl` from the unix prompt will bring you to the `zctl` prompt, which looks like this:

```
zctl:
```

You can now change your subscriptions list.

Each subscription consists of three pieces of information: the class, the instance, and the recipient. The class tells Zephyr what type of information is being transfered. Right now we're only interested in messages, which have class **message**. Note that class, instance and recipient strings are case insensitive (i.e., `zippy` will be treated the same as `Zippy` or `ZIPPY`).

The recipient can be either `%me%` or `*`. Personal messages (like `How about lunch?`) are sent to instance `personal` and recipient `%me%`. These are already subscribed to when you start using Zephyr. For all other instances[4], the recipient is `*`, meaning "everybody."

So, to add a subscription to all the `presidents` messages, type

```
zctl:  add message presidents *
```

at the zctl prompt, and you'll start getting those messages. If you later want to stop getting them, type

```
zctl:  delete message presidents *
```

You can invent an instance of your own. Just tell people to `add` it, then use `zwrite -i foo` and you have your own instance! Many fraternities and dorms have their own. Ask around.

It should be noted that it is possible to subscribe to *all* instances in class **message** (the default), and there are a number of people who do; this means that you should *never* consider an instance in class **message** to be private. If you invent a new class (simply substitute the name of the class, anything you want, for "message" in the commands discussed here) it is exactly as private as the name is: the only way to keep people from subscribing to a class is to keep them from knowing the name.

You can also try out an instance by using the zctl commands `sub` and `unsub` instead of `add` and `delete`, like:

```
zctl: sub message presidents *
```

and

```
zctl: unsub message presidents *
```

[4]Well, not really. There are also *urgent* messages.

If you're lazy (or nosy) and just want to see *everybody's* public messages, you can `add message * *`, and you'll get messages from every instance. If you're getting deluged, you can always return to simpler days by doing `delete message * *`.

If you decide you still want to be subbed to every instance in class message except the instance `silly_twits`, you can use the *zpunt* program. (You must add the sipb locker to do this, if you haven't before.) At your athena prompt, simply type

```
zpunt silly_twits
```

When you're done with `zctl`, type `quit`.

Aargh! I have three papers due tomorrow and everyone's zwriting me! Help!

If you want to hide for a while, you can type `zctl hide` from the Unix prompt, or just `hide` from the zctl prompt, and it will look like you're not logged in. However, zephyrs sent to you *will* reach you. When you're ready to face the world again, `zctl unhide` will restore your place. You can also type `zctl wg_shutdown`. This will prevent *any* messages from reaching you. `zctl wg_startup` will let you receive messages again.

How can I tell if someone's logged in?

There are two ways to find this out. If you just want to know where Jack Florey is now, you can type

```
% zlocate jflorey
```

and Zephyr will reply with his current location (e.g., `M11-116-1`), if he's logged in and not hidden, along with the date and time he logged in there.

If you want to be told whenever a friend logs in or out, you can use `znol`, the "Zephyr Notify On Login" program. You simply create a file in your account called `.anyone`, and put your friends' names in it, one to a line, like this:

```
jtkirk
spock
lmccoy
uhura
jrand
```

Then, each time you log in, type `znol`. (You can add this to your `.startup.X` file, too.) You'll get a list of everyone in your `.anyone` file who's currently logged in; then, whenever someone logs in or out, you'll get a Zephyr notice.

Actually, that's not quite true. The person has to have told Zephyr, "Whenever I log in, announce my arrival." This is done by typing

```
% zctl set exposure net-announced
```

once. (It is automatic after that.) There are several other choices besides `net-announced`. The first of these choices is `none`. This disables Zephyr entirely. Next is `opstaff`, which allows you to receive important/emergency messages about the status of Project Athena. Another, `realm-visible`, will let people `zlocate` you, but it will not send messages when you log in and out. The default is `realm-visible`.

How do I tell people I'm not at the terminal?

If you want to leave the terminal, and want an automated response to be sent to anyone who zephyrs you, you can use `zaway`. Simply type `zaway` at your `athena%` prompt, and walk away. Anyone who zephyrs you will get the following message:

`I'm sorry, but I am currently away from the terminal and am not able to receive your message.`

For more information on zaway – for instance, how to send a `zaway` message to only one person, or how to send different messages to different people, read the zaway manpage (type `man zaway` at your athena prompt).

How do I customize Zephyr?

Obviously, we've only been able to give you enough information to get you started. There are more options in `znol`, more commands in `zctl`, and more programs (like `xznol` and `xzwrite`) than can be covered in a few pages. If you're interested, look at the manual pages for each of these (with `man xzwrite`, for example).

Most Zephyr customization is done with the *.zwgc.desc* file, which `zwgc` uses to describe how the windowgrams, as they are often called, look on the screen. You can learn a lot by looking at the *.zwgc.desc* files of other users, as well as the standard file */usr/athena/lib/zephyr/zwgc.desc*, which has explanatory comments.

Additionally, the SIPB typically gives a *Zephyr Hacks* minicourse during IAP. If you're interested, come by our office and let us know.

An Introduction to Discuss

What is Discuss?

Discuss is a networked, electronic conferencing system. It is similar to bulletin-boards that have become popular on microcomputers, except that it is accessible from time-sharing machines and workstations on the network. Discuss allows a user to attend a set of electronic meetings. Each meeting has a specific focus, such as a project, class, or special-interest.

Users may select ("attend") meetings, and read messages ("transactions") that have been entered in those meetings. A user may choose to enter new transactions into a meeting, either by replying to a previous one or by starting one on a new topic.

Additional information regarding Discuss can be found in *An Inessential Guide to Discuss*, also published by the SIPB.

Keeping track of meetings

Discuss does not keep a central list of all Discuss meetings. Instead, it maintains a list for each user of the meetings that they attend. To specify a meeting you can use the meeting name, or your own abbreviation. In the future it may be possible to use another user's abbreviation by specifying the username in the command line. This approach allows users to have their own names for meetings. For example, the "Discuss_development" meeting could be known as "discuss" to one user, "dsc" to another, or both to yet another user. All this information, as well as the information needed by the client programs, is stored in a file called *.meetings* in your home directory. Without this file, Discuss will not be able to locate meetings.

Using Discuss

Initial setup The best way to understand the ideas above is to start playing with the system. They should become clear after a little practice. Here is a step-by-step explanation of how to use Discuss.

First, run the Discuss run. To run it, type `discuss` to your prompt. The first thing it will do is look for a *~/.meetings* file. If you do not have one, Discuss will assume this is your first time running it. It will then ask you if you want it to create a *.meetings* file. This will create a prototype *.meetings* file in your home directory for you to start with.

Running Discuss without a *.meetings* file looks like this:

```
% discuss
discuss: No such file or directory

If you are using discuss for the first time, or if you have only used the
experimental version of discuss, you need to run the 'dsc_setup'
```

```
command from the shell.

Run dsc_setup now? (y or n)  y

Running setup...
  Creating .meetings file:
  done.
discuss:
```

The version of this program distributed on Athena will give you a *.meetings* file that lists two meetings, called "New_Meetings" and "Everybody". The former is for announcements of new meetings, and the latter is for random announcements, questions, and anything else that may be of general interest.

Finding interesting meetings

The meetings you currently attend are listed in your *.meetings* file. To list the meetings in your *.meetings* file type *list_meetings*.

The transactions in "New_Meetings" can be used to add the meetings you wish to "attend" to your personal list of meetings.

To attend the New_Meetings meeting, simply *goto* it.

```
discuss:  goto new_meetings
New_Meetings meeting;  175 new, 175 last.

discuss:
```

To learn more about a meeting listed in "New_Meetings," *print* the corresponding transaction (This transaction is usually identical to the first transaction in the announced meeting). For instance, you could do the following:

```
discuss:  print 163
[0163] srz@LCS.MIT.EDU  New_Meetings  04/01/88 15:17 (7 lines)
Subject: Central_America meeting
  Meeting Name:  Central_America
  Host:          bloom-beacon.mit.edu
  Pathname:      /usr/spool/discuss/ca
  Participation: Public

This meeting is an Athena-wide "panama" system.  This meeting is
fed nightly with all the changed .plan files of all Athena users.
--[0163]--
```

```
^L

discuss:
```

If you want to read the transactions in Central_America you need to add it to your personal list of meetings. To do this use the command *add_meeting* (*am* for short), giving it the transaction number of the announcement as an argument:

```
discuss:  add_meeting 163
Transaction [0163] Meeting Central_America (ca) added.

discuss:
```

Saying something

There are two ways you can enter a new transaction in a meeting. To start a new chain, use the *talk* command (*t* for short), *enter*, or *send*. Discuss will ask for a subject which will be printed at the head of the message. It will then ask you to enter the text of the message – a `C-d` or a "." on a line by itself is used to signal the end of the text. If you prefer to use an editor, you can type, or if you realize at any point while entering the message that you want to edit it, type `~e<RETURN>` (Yes, that's a twiddle, then an 'e', then the RETURN or ENTER key.) on a line by itself, and Discuss will start up the editor defined in your EDITOR environment variable, with whatever text you have already entered already entered into the editor. Alternatively, if you prefer to use an editor most of the time, you can use `setenv` to set a variable called DISCUSS_EDITOR[5] to the pathname of whatever editor you prefer to use, and it will be called automatically each time you enter a transaction. In this case, if you wish to enter a message without using the editor, you enter *talk -no_editor*.

The other way of entering a transaction is to reply to an existing transaction. For this, use the *reply* command (*rp* for short). This will start up the editor as with *talk*. The transaction will be entered as a response to the current transaction (by default), or the transaction specified.

More help

See the document **Using Discuss**, available through the SIPB, for more complete and detailed information about discuss.

Credits

Many thanks to Stan Zanarotti, Ken Raeburn, and Nancy Gilman, writers of **Using Discuss**, from which this section was fabricated.

[5]You can put this command in your `.environment` file.

Usenet: Talking to the world without paying long-distance bills

The Usenet...

The Usenet (Users' Network) is a bulletin board shared among many computer systems around the world. Usenet is a logical network, sitting on top of several physical networks, including UUCP, BITNET, and the Internet. Sites on Usenet include many universities, private companies, and research organizations.

Before using or learning to use the Usenet, it is helpful to understand some of the terminology used to describe it:

newsgroup: A section of the net in which one particular topic is discussed. Newsgroups are named using a hierarchical tree structure quite similar in nature to the Unix™ file structure, but the net uses periods while Unix™ uses slashes. For example, discussion centering around the Apple Macintosh computer appears in a newsgroup entitled comp.sys.mac, while discussion about politics in general appears in talk.politics.misc.

article: A message which someone has posted to the net. Other terms for an article include "message" and "post."

subscribe: In order to read a newsgroup, you subscribe to it. If you decide that you no longer wish to read the newsgroup, you unsubscribe from it. By default, you are asked for each new newsgroup if you want to subscribe to it.[6]

`.newsrc` **file:** Your .newsrc file resides in your home directory and tells `rn` which newsgroups you know about, to which ones you are subscribed and unsubscribed, and which articles in each newsgroup you have read.

`KILL` **files:** Despite the fact that the net usually tries to limit newsgroups in breadth of topics, you will still always find articles that you don't want to read in newsgroups that you DO want to read. `rn` stores a `KILL` file for each newsgroup, and these files instruct the program to get rid of certain articles before you even have to look at them. To "kill" (or "junk") an article is to make the system think you've already read it.

Using `rn` on Athena.

The SIPB currently supports several different interfaces to the Usenet[7], but in this document we'll center on `rn`. To use `rn` you must add the `sipb` locker by entering the command `add sipb` at your Athena prompt. You can put this command in your `.environment` file to cause it to occur when you log in.

`rn` operates on three levels: the newsgroup selection level, the article selection level, and the paging level. Each level has its own set of commands, and its own help menu. At the *newsgroup*

[6] This is the default for `rn`, the program we'll be using here. The other programs have other defaults.

[7] Other common programs are `xrn`, an X-based interface, and `GNUS`, a package for reading news within GNU Emacs. For more information on these, look at the on-line documentation, or come by and ask us.

selection level (the top level), you may specify which newsgroup you want next, or read them in the default order, which is the order that the newsgroups occur in your `.newsrc` file. (You will probably want to rearrange your `.newsrc` file to put the most interesting newsgroups first. This can be done with the 'm' command on the newsgroup selection level, or by using a file editor such as `emacs` to edit the file directly.) At the article selection level, you may specify which article you want next, or read them in the default order, which is either in order of arrival on your system, or by subject threads. At the paging level (the bottom level), `rn` behaves much like the `more` program.

The most important command to remember in `rn` is 'h', for help. On any level, at any prompt, an 'h' may be typed for a list of available commands. Furthermore, typing space to any question means to "do the normal thing." You will know what that is because every prompt has a list of several plausible commands enclosed in square brackets. The first command in the list is the one which will be executed if you type a space.

Some of the more important commands besides 'h' and 'space' are:

ALL LEVELS:

`u` will unsubscribe from the current newsgroup.

`c` will "catch up" on the current newsgroup by marking all the messages in it read.

`C-k` will allow you to edit the KILL file for the current newsgroup.

NEWSGROUP LEVEL:

`g` *newsgroup* will go to the named newsgroup. If it is unsubscribed, the user will be asked if he or she wants to resubscribe to it.

ARTICLE OR PAGER LEVEL:

number will go to the message with the given number.

`k` will kill all unread articles with the same subject line as the current article.

`K` will kill all unread articles with the same subject line as the current article and deposit this kill command in the KILL file for the current newsgroup so that it will be executed each time the newsgroup is accessed.

`f` and `F` will post a follow-up article to the current article. If an upper-case F is used, the text of the current article will be embedded in the follow-up.

`r` and `R` will respond to the current article through net mail. If an upper-case R is used, the text of the current article will be embedded in the mail letter.

`$` will go to the end of the current newsgroup. Using the 'r,' 'R,' 'f' or 'F' after using the '$' command command will allow you to post an article or write a mail letter from scratch.

These are only a few of the commands available in `rn`. Rather then attempting to summarize everything that can be done with `rn`, we recommend that you print out the manual page for the program and read it over. While it is not necessary for you to read the manual page to use `rn`, it might be useful.

If you just wish to view the man page on screen, issue the command

```
% man rn
```

Even if you choose not to read the man page now, it is a good idea to have it with you as a reference when you are first using `rn`.

Hints and tips

news.announce.newusers: It is a good idea to read through news.announce.newusers before unsubscribing it. If you choose not to read the entire newsgroup, you should at least read the articles containing answers to frequently-asked questions and the Usenet Primer. Use `g news.announce.newusers` from the newsgroup level, then '=' to list the subjects, then enter a number to read the articles with the subjects you wish to read.

Using a `.signature` file: `rn` and `mh` disagree over what a `.signature` file should do. While `rrn` thinks it is something that should be placed at the end of a file, `mh` thinks that it should be placed in the "From:" field of a mail message. Therefore, if you want to use a `.signature` file with `rn`, you will have to set a variable called "Signature: " in your `~/.mh_profile` file. Otherwise, `mh` with look at your news posting signature, and do the wrong thing with it. Simply edit your `.mh_profile` and add a line like

```
Signature: Joe Random User
```

to the end. You can then put a signature in `~/.signature` to be automatically added to the end of your news postings.

Including an article in your follow-up: When you use the 'F' command to post a follow-up article which contains the text of the original article, `Pnews`, the program which handles such posting, gets very picky about how much new text you put into your article compared to how much you retain from the one to which you are responding. In most cases, you will edit the article you are including so that you only include the parts relevant to your followup (but be sure not to disrupt the `References:` line in the header section.). But if you attempt to post an article which has more lines from the last article than lines you have added to it, `Pnews` will reject the article. The way to avoid this is to fool `Pnews` into thinking that lines from the original article are actually lines that you posted. Since the lines from the original article begin with '>' by default, replacing the '>' with another character fools `Pnews`. The command sequence within emacs to do this is: `M-x` *replace-string* `<RETURN> C-q C-j > <RETURN> C-q C-j` *string* `<RETURN>`, where *string* is the character (or characters) with which you want to replace the '>'.

AFS: The Andrew Filesystem

Any particular method for handling files of data is called a *filesystem.* Athena currently uses *AFS*, the *Andrew Filesystem*[8], as the filesystem for all user home directories and most of the other lockers.

AFS is a *distributed* filesystem. This means that the files stored using *AFS* are located on a remote server, so you can access them from any Athena workstation. In fact, the owner can make the files available from any computer running *AFS* in the whole world.

You don't have to attach *AFS* filesystems[9]; they are always available from the workstation under the directory */afs.* Another major difference between *AFS* and other distributed filesystems is that groups of *AFS* files can be replicated onto multiple servers in such a way that makes the failure of one server unnoticeable to a user. In addition, files and directory structures are cached locally to minimize network traffic to the server. This makes access to *AFS* files very fast after the first time a file is read.

AFS Volume? Why do I care how loud my files are?

No, no, no; that's not what *AFS* volume means. In order to understand *AFS*, it is important to know some basic terminology. Following is a list of terms to provide you with some important background for when you use *AFS*:

Volume: An *AFS* volume is a collection of files and directories that are grouped together as one unit. A volume can be backed up, renamed, moved from one server to another, replicated, created, or destroyed as a unit.

User: A user is someone who is registered with *AFS*. You must be registered with *AFS* if you wish to be able to own files. The process of telling *AFS* who are you is called *authentication.*

Group: A group is a list of users. Currently, it is not possible for a group to contain other groups.

Cell: An *AFS* cell is essentially a domain of authority. It is currently the largest unit used in *AFS*. Each cell has a list of administrators, a list of users, a list of groups, and a list of volumes. Each of these lists is private to the cell. This means that an administrator in one cell may or may not have any access to another cell. A user in one cell may not even exist in another. A cell is also a domain of authentication. If you are the user *qjb* in the *athena.mit.edu* cell, you may not have any access at all to files belonging to a user named *qjb* in the *andrew.cmu.edu* cell. Likewise, someone who has administrative capabilities in the *sipb.mit.edu* cell may not be able to touch the *athena.mit.edu* cell.

ACL: ACL stands for Access Control List. When referring to access control lists, some people say the initials ("A-C-L") and others pronounce the acronym ("ackle"). An access control

[8] *AFS*was developed at Carnegie Mellon's *Project Andrew*, which is a large-scale distributed workstation experiment similar to *Project Athena*, the predecessor of today's *Athena*. It has since become a commercial package sold by *Transarc Corporation.*

[9] You might want to use the `attach` command for Athena file systems anyway – this is discussed later.

list pairs users and groups with operations that they are allowed (or not allowed) to perform. Although access control lists are used for much more than just *AFS*, only *AFS* access control lists will be discussed here. A typical access control list would say something like, "The user *potato* is allowed to create files in this directory. The user *soup* is allowed to modify existing files and create new files in this directory. Anyone else is allowed only to read files in this directory." *AFS* access control lists and how to use them are explained in more detail later in this document.

How do I use *AFS*?

In general, you won't even notice whether files you are using come from *AFS* or not. The time when it really does make a difference, however, is when you want to change file protections, and this is critical since your Athena home directory is in *AFS* space.

If you wish to list or change protections, you will need to become somewhat familiar with the program `fs`. This program allows you to get various kinds of information about *AFS* files and directories and to change and look up protections on directories. Although it is not necessary for you to understand standard UNIX file protections in order to understand *AFS* file protections, it will probably help you if you do. Standard UNIX file protections are explained in the **File Protection** section of this document.

The `fs` command has a built-in help feature. Typing `fs help` will give you a lot of information on how to use various subcommands. All the commands that you will probably want to use, though, are described here.

Authenticating to *AFS*

Before *AFS* can know who you are, you have to be *authenticated* to the cell. Authentication is just the name given to confirming your identity. There are several ways to authenticate to *AFS*. The most common way will be through the `attach` command. If you attach a filesystem that is in *AFS*, you will be authenticated automatically to it. Should you have to reauthenticate for some reason, you can just reattach whatever it is that you are trying to use.

`attach` authenticates to *AFS* by using a program called `aklog`. Generally, typing `aklog` with no arguments will reauthenticate you to whatever you are using. To be safe, however, you can specify the path of the file you need to reach. For example, if you are working in the *potato* locker, you can type `aklog /mit/potato` to reauthenticate.

File protections under *AFS*

AFS offers considerably more flexibility in directory protections than regular UNIX. Unfortunately, though, protections currently can be set on files only in a limited way. Protections in *AFS* are enforced with access control lists instead of with bit fields as in regular UNIX. This means that in *AFS* it is possible, for example, to give one set of users read access to a directory, one set of users write access, and another set no access at all.

There are seven types of access that can be granted:

Read (r): Read access on a directory implies permission to read the contents of all the files in that directory. It has no implications about access to subdirectories.

Lookup (l): With lookup access on a directory, it is possible to look at the directory's access control list and to list of the contents of the directory (*i.e.*, what files and directories are in it). It does not imply read access to the files. **If you do not have lookup access to a directory, no other form of access can be used.**

Insert (i): Insert access on a directory implies permission to create files or subdirectories in the directory. It does not imply the ability to modify the files that are created, however. Insert access without write access is useful mainly for the case when you want to allow someone to create subdirectories in a given directory but not to modify files that are already there.

Delete (d): Delete access on a directory implies the ability to remove files or empty subdirectories from the directory. Like insert, delete access does not imply write access.

Write (w): Write access on a directory grants permission to modify files and subdirectories within a directory. It implies neither insert nor delete access to the directory.

Lock (k): A user with lock access on a directory can put advisory locks on files within the directory. This is typically useful only to applications programmers.

Administer (a): With administer access on a directory, it is possible to change the access control list of the directory. Administer access does not imply any other kind of access, and like all other types, is useless without lookup access. Administer access, contrary to some people's intuition, cannot be "forced" onto subdirectories. It only applies to the directory on which it is set, not that directory's subdirectories.

In addition, the *user* portion of a files UNIX mode bits (described in the **File Protections** section of this document) apply to all users, not just the owner of a file. That means that if a file's *user* mode bits show the file to be unreadable, the file will not be readable by anyone regardless of the access control list set on its parent directory. Likewise, if the *user* mode bits are set to give *read*, *write*, and *execute* access, then access to the file is determined completely by the *AFS* access control list on the directory. The *group* and *other* bits as well as the *UNIX group* of the file are currently ignored in *AFS*[10]. If you didn't understand that, ignore it.

The owner of a volume has automatic administer access to all directories in the volume. The owner of a directory has automatic administer rights to that directory.

Access can be granted to individual users or to groups. There are two special groups: *system:anyuser* and *system:authuser*. *system:anyuser* means all users whether or not they are known

[10]Well, actually, the *UNIX group* field and *group* mode bits are used in a limited way for *setgid* executables, but this is beyond the scope of this document.

to the cell. *system:authuser* means any user who is registered with the cell and has been authenticated (*i.e.*, anyone who has told *AFS* who they are).

Other groups can be one of two types: *system* groups or *user* groups. System groups have names that start with *system:*. They are controlled by the system administrators. User groups start with <*user*>*:*, where <*user*> is the name of the user who created the group. It is possible for any user to create his or her own groups in *AFS*. Information about how to do this appears below.

An access control list on a directory contains a list of users and groups and the operations they are allowed to perform. To find out what the access control list on a directory is, you can use the command `fs listacl <directory>` where <*directory*> is the name of the directory you are interested in. Note that `fs la` is short for `fs listacl`.

For example:

```
athena% attach sipb
sipb: AFS mounted /afs/sipb.mit.edu/project/sipb on /mit/sipb (read-only)
athena% fs la /mit/sipb
Access list for /mit/sipb is
Normal rights:
  system:gsipbbin rlidwka
  system:anyuser rl
```

This means that members of the system-controlled group *system:gsipbbin* have full access to */mit/sipb* and all other users have read and lookup access only.

Here is another example:

```
athena% attach qjb
qjb: AFS mounted /afs/sipb.mit.edu/user/qjb on /mit/qjb (read-write)
athena% fs la /mit/qjb
Access list for /mit/qjb is
Normal rights:
  qjb:plorq rla
  qjb:all rlidwka
  system:anyuser rl
```

This means that the user-controlled group *qjb:all* has all access to */mit/qjb*, the user-controlled group *qjb:plorq* has read, lookup, and administer access to */mit/qjb*, and everyone else has read and lookup access only.

Where to go for more

The SIPB also publishes a guide called **Inessential AFS**. This document contains everything you see here and some additional information about changing access control list, quotas, and other *AFS* programs. Good luck!

Lost Contact With File Server

At some point in your dealings with Athena, you will almost certainly encounter a message saying something like:

```
afs:  Lost contact with file server 18.70.0.210 in cell athena.mit.edu
```

(The number may be different.) This is disconcerting if you've never seen it before. But DON'T PANIC.

What This Means To You

(The number may be different.) This means that your workstation has lost contact with a *file server* that your workstation is using. You may be unable to use some programs (depending on what file server you've lost, this can be anything from your background to *idraw* to *ls*); you may be unable to save or retrieve your files. If you've gotten this message, can't save your files, and only want to know how to save and/or print your files, you can skip to the end. If you want to know why it's happening and what you can do, read the next section first.

What Did All That Mean?

Files in the Athena system are kept in *lockers* on *file servers.* A locker is basically a chunk of space on a disk somewhere, generally in building E40 or W20 or 11; a file server is a workstation, exactly like the workstations you'll log in on, but with rather more disk space. If a file server goes down – for maintainance, or there's a power outage, or terrorists take over building E40 and refuse to turn the machines on until President Vest sings the theme to Gilligan's Island – you will not be able to access the files on that file server until it comes back up. Shortly after a servers comes back up you will see a message saying this:

```
afs:  file server 18.70.0.210 in cell athena.mit.edu is back up
```

If you have attached a locker (the sipb locker, for instance, or the graphics locker) and the file server that contains that locker goes down, you will get a "lost contact" message in your console window. Some lockers are spread out over more than one file server so that if one goes down you will still be able to access the information. For more information about this, you can look in the *AFS* section of *Inessential Athena.*

So What Did I Lose?

The first thing you want to do will probably be to find out which locker you've just lost. Suppose you have just received the message:

```
afs:  Lost contact with file server 18.70.0.210 in cell athena.mit.edu
```

The number 18.70.0.210 is the internet address of the machine; the command `hostinfo` will tell you what the name of the machine is. So typing `hostinfo 18.70.0.210` will give you this:

> Desired host: 18.70.0.210
> Official name: ROSEBUD.MIT.EDU
> Host address: 18.70.0.210

This tells you that the name of the machine is *rosebud*. If you want to know whether a particular locker is on that server, you can use the command `lookup` (in the consult locker). Thus `lookup sorokin` will return:

Filesystem sorokin (AFS) is located on:

> ROSEMARY.MIT.EDU (read-write)

which says that sorokin's homedir is still accessible. `lookup sipb` will return:

Filesystem (AFS) is located on:

> ROSEBUD.MIT.EDU (read-write)
> ROSEBUD.MIT.EDU (read-only)
> RONALD-ANN.MIT.EDU (read-only, alternate)

This tells you that sipb is located on rosebud. Handily, there is another file server containing the same data (ronald-ann), and so you probably shouldn't be worried. However, you may want to look at *all* the filesystems you have attached. The command for this (also in the consult locker) is `wherefs`. It will give you something like this:

> sorokin : AFS filesystem on host ROSEMARY.MIT.EDU
> sipb : AFS filesystem on host ROSEBUD.MIT.EDU
> gnu : AFS filesystem on host ARTEMIS.MIT.EDU
> consult : AFS filesystem on host CHICKADEE.MIT.EDU
> graphics : AFS filesystem on host KITE.MIT.EDU

There may be lockers listed that you don't recognize. The reason for this is that when you attach a locker, it is connected to the workstation. Therefore, a locker attached by the previous user may show up in your listing. You can ignore it.

So What Should I Worry About?

First of all, servers will often come back up fairly quickly. If you have the time and something to do, it may be worthwhile to just wait.

If the locker on the lost server is not one you're using, or is replicated elsewhere, you can pretty much ignore the message.

If you need a program on that server, you can try to find out how long the server will be down or whether that program exists anywhere else. (Try asking on instance *help*; see the section on zephyr for details on instances.) There may be nothing you can do but go home.

But I Can't Save My Files!

First of all, don't assume that not being able to save your files means that you've lost your fileserver. Make sure that this *is* the problem.

If it is the case, you can still save your files. You just need to save them somewhere else. In emacs, the command *C-x C-w* will present you with a pathname, and will ask you to type in the name of the file you want to save to. Delete all of the pathname that it gives you, and type `/usr/tmp/`*filename*. In EZ, simply choose the *Save As* option. You have now saved your file in a temporary file on the hard drive of your workstation. This file may hang around for up to three days. You can mail it to yourself, with the command

`/bin/mail` *username* `<` *filename*

DON'T INCORPORATE IT NOW. WAIT UNTIL YOU HAVE YOUR HOME DIRECTORY BACK.

Now, disaster averted, you can wait for the fileserver to come back up, or you can come back later. While the server is down, you will not be able to access any of your files.

Command Not Found

If just about every command returns *Command not found,* you have probably lost your *system packs.* Since Athena workstations need to have available more software than will normally fit on the hard disk drives of all the machines, and since keeping separate copies on each workstation would waste useable disk space and make software upgrades more difficult, a large part of the 'standard' software is actually kept on remote filesystems called *system packs* or *syspacks.* This happens very, very, very rarely, but if it does you may well want to give up and go home. Eat lunch, take a nap, sit outside if it's nice out. Until the server comes back up, you can't do anything. Sorry.

Useful Lockers

The following is a partial list of useful lockers. To access them, type `add` *filesystem*. Many of them have a file named README in their top-level directory, which will have more information about the filesystem, its layout, and whom to contact for more information.

gnu contains software written and distributed by the Free Software Foundation, such as gcc, a C/C++/Objective C compiler; gdb, a source-level debugger; gnuchess, a chess program; bison, a replacement for YACC; and gas, an assembler. These programs are available in varying degrees for various platforms. Send any problems or questions to *gnu@athena.mit.edu.*

consult contains software written and/or maintained by Athena's MIT-IS-CSS[11] Consultants. The *consult* locker contains programs for the vax, rt, decmips, RS/6000, NeXT, and Sun4 machines. Read */mit/consult/README* for instructions to use or learn about software contained in *consult.* Send any problems or questions to *consult@athena.mit.edu.*

games is a random collection of games and puzzles. Currently, the games locker supports games for vax, rt, decmips, and RS/6000 to varying degrees, in addition to games to be run within Emacs. The locker also contains source code for most of the games that are installed. Look at the */mit/games/POLICY* file in the games locker for rules about submissions. Send any problems or questions to *bug-games@athena.mit.edu.*

info contains up-to-date information about the status of Athena software and hardware, such as notices of impending or emergency machine shutdowns. Comments should be sent to *bugs@athena.mit.edu.*

net-directory contains the network directory produced at the University of Texas, as well as other information about the Internet network. See the *README* file in */mit/net-directory/README* for more information.

watchmaker is maintained by the so-called "watchmakers,"[12] a group of student systems programmers at Athena. **BEWARE!** Most of the programs in the locker are esoteric and only useful to experienced systems hackers, but some of the programs are of general interest. Report any problems or bugs to *watchmakers@athena.mit.edu.*

graphics The graphics locker contains various graphics-related programs useful for viewing and manipulating graphics. See the **Graphics** section of this guide for more information.

postscript is a collection of interesting and useful PostScript programs and utilities. See the *discuss* meeting on postscript_hacks for information about what is currently stored in the filesystem[13]. PostScript is a page description language developed by Adobe Systems Inc. and is available on all of Athena's printers. It is a public locker, with world write access turned on, so that anybody may place copies of useful PostScript code into it for central storage. Problems should be sent to the discuss meeting above.

[11] That's Massachusetts Institute of Technology Information Systems Computer Support Services. Sigh.

[12] Read *The Mote in God's Eye* by Larry Niven and Jerry Pournelle for an explanation of this term.

[13] Discuss is a conferencing system; more details are in the **Discuss** section or in the **Using Discuss** document, distributed by the SIPB

perl Perl is an interpreted language that fills in the area between shell programming and C programming. The locker contains both source and binaries, and various sample programs. If you're interested, you should also look in /afs/sipb.mit.edu/contrib/perl [14] a collection point for perl contributions.

rfc contains all the current Defense Advanced Research Projects Agency (DARPA) Requests For Comment (RFC's). These server are documents on "standards". It also has an index, */mit/rfc/rfc-index.txt*, which you should definitely read, because the proposals are listed only by number.

frame FrameMaker is an excellent but mostly untested publishing tool, which handles word processing, books, page layout, and graphics. You are encouraged to play around with it; it is a complicated program, and very very new (as of August, 1993, anyway), but will be supported by Athena. Documentation is being written even as you read this, unless it's already done, in which case it will be available from Graphic Arts. Look at the README file in */mit/frame/README* for more information.

dotfiles The dotfiles locker contains a number of extensively modified dotfiles. Many of these may be useful, and there are a number of neat hacks that you can play with; but be *sure* you know what you are doing before you hack your own dotfiles. Read the README file in */mit/dotfiles/README* for an explanation of how the locker is set up.

Some other neat lockers are *shakespeare, bible* and *weather*. They're just about exactly what you think they are. Look around in them; ask your friends for their suggestions; see what you can find on your own.

[14] Look at either the **AFS** section of this guide, or in the SIPB document *Inessential AFS*.

Kerberos: Athena's Watchdog

Kerberos; also spelled *Cerberus.* *"n. The watch dog of Hades, whose duty it was to guard the entrance–against whom or what does not clearly appear;...is known to have had three heads..."*
-Ambrose Bierce, *The Enlarged Devil's Dictionary*

Kerberos is the system designed by MIT to provide security on the Athena network. This document will not provide you with a complete description of the mechanisms and the math used by *Kerberos*; it will explain the necessity for *Kerberos*, and will quickly summarize what *Kerberos* does. For more information, you can look at the Kerberos section of the Athena Technical Plan, the PostScript file */mit/kerberos/doc/techplan/techplan.PS.*

Why *Kerberos* is needed. *Kerberos* has two purposes: security and authentication. On most computer systems, a password is used to prove a user's identity; on a distributed network system, like Athena, this password must be transmitted over the network, from the workstation being used, to any other machines containing files or programs the user wants access to. Because this password is the one secret piece of information that identifies a user, anyone knowing a user's password can effectively *be* that user on Athena, reading their files, sending mail as that user, etc. **Please note that THE ELECTRONIC COMMUNICATION PRIVACY ACT of 1988 does make this a Federal crime** punishable by all kinds of nasty stuff you don't want to be punished by. No kidding.

Obviously, it is therefore necessary to prevent anyone from intercepting or eavesdropping on the transmitted password. In addition, it is necessary to provide a means of *authenticating* users: any time a user requests a *service,* such as mail, they must prove their identity. This is done with *Kerberos*, and this is why you get your mail and no one else's.

A Few Handy Definitions We have now used two pieces of jargon in one paragraph, and at this point you will probably want to learn the meanings of a few terms that will be used in this section.

user: A person using a computer system. A user, through her workstation, may make a series of requests to several servers. This user, we assume, would like to avoid retyping her password every time she makes such a request.

service: Very simply, a service is a program or set of programs, running on a computer which is accessible over the network. A user will request a service for the workstation which she is using; the service will want to be sure that the service is really being used by that user.

principal: A principal is some entity which can prove its own identity and verify the identity of other principals. Each **user** and each **service** registered with *Kerberos* is a principal, since *Kerberos* provides the authentication services required.

ticket: Once a user has proved her identity to *Kerberos* with her password, *Kerberos* sends a block of encoded data, called a ticket, to the user. It is this ticket that is used to prove a user's identity to a service. Tickets are stored in the */tmp/* directory and are erased upon logout. (You will get a message in your console window that says *Tickets destroyed.*) Tickets will expire after 10 hours.

authenticator: When a user tries to use a *service*, her workstation sends an block of data called an *authenticator*, built from the *Kerberos* ticket and containing a timestamp and the name of the workstation, to that service. The service decodes it, verifies that the user is who she says she is, and then lets that user, at that workstation, use the service.

This is the basics of how *Kerberos* works – for the most part, you'll see none of it, which is a nice feature. For a longer and more technical explantion, you can look in:

`/mit/kerberos/doc/techplan`

Once registered with *Kerberos*, tickets are obtained by the login program every time you log into a workstation. You can also manually obtain new tickets (which you usually do only if your old ones have expired, 10 hours after you log in) by running the program `renew`. You can also use`kinit`, which prompts for a username, requests an initial ticket from *Kerberos*, and then asks for your password. If you are not registered with *Kerberos*, it will print `Principal unknown (Kerberos)`. Unless you mistype your username, this should not happen. To correct this, or any other errors, contact a Consultant or the Athena Accounts Administrator.

Moira: Athena glue-all, or Your Fate

As you may have gathered by this time, you, as a user of Project Athena, are aided by a fairly large number of services. You are served by *NFS* and *AFS* which bring you your files, by *Kerberos* which allows you to convince Athena that you are who you say you are, by *Hesiod* which allows you to look up information about many things, by *Zephyr* which allows you to locate users and communicate with them electronically, by the post offices which allow you to send mail – the list goes on. It would be virtually impossible for a single person to keep all of these services under control and consistent with each other. This is why we have *Moira*, the Service Management System. *Moira* is an alternate spelling for *moera*, which means fate. The three aspects of fate were Clotho, who spun the thread of life; Lachesis, who wove the thread into the tapestry of life setting a person's destiny; and Atropos, who cut the threads at the appropriate time. Similarly, *Moira*, the system, creates user accounts, controls the user's home placement and quota, and deactivates user accounts[15].

Moira is a very large database containing most of the information that is necessary to keep Athena running smoothly. Although *Moira* is not needed for someone to log in and use the system (unlike *Kerberos*, which must be available all the time), without it, nobody would be able to change anything about any service. This means you couldn't add anyone to a mailing list, create a new filesystem, or any number of similar operations.

Even though *Moira* controls a vast array of things and can perform several tasks, there are only two things that you will probably ever need to use *Moira* for: changing or inspecting lists and updating finger information. Both of these are explained in this section.

Things Not To Do

It is possible for you to put yourself into a situation that you won't be able to get out of. For example, you can change the administrator of a list you own to someone else. If you do this, **you will not be able to reclaim the list without help from an administrator**. When you make a change, especially a transfer of authority, think about whether you'll be able to undo it. If you really get messed up, you can call the user accounts consultant at x3-1325.

How to manage lists

To *Moira*, a list is simply, well, a list. The items on an *Moira* list are called *members*. Each member of an *Moira* list has two attributes: a type and a name. A *Moira* list can have several types of members. The only ones useful to most users are users, other lists, and arbitrary strings such as electronic mail addresses. When you registered for your account, you were given a list containing one member: you, the user. If your username were *juser*, your list would be called *juser* and would have one member: *USER juser*. A list has other properties besides members. The most important of these is the administrator. **If you registered for your account before September, 1988,**

[15]Until the beginning of 1989, *Moira* was known as *SMS*, which stands for *Service Management System*. Since so many other things were called *SMS*, and since almost everything else at Athena had a greek name, it was decided that the Service Management System should be renamed.

your list may be its own administrator. This means that any member of the list could add members, or even delete you from the list! This section will show you how to correct this.

Lists can serve several functions. The two most common are UNIX groups and mailing lists. Your list is a *Unix group*. You use it to control file access. For more information about Unix groups, look at the manual pages for `chgrp` (change group), and `chmod` (change mode) by typing `man chgrp` and `man chmod`. You can also find information in this guide in the **File Protection** section.

The main tool that you will use to manipulate lists is `listmaint`. `Listmaint` is a fairly self-explanatory, menu-based program. To run `listmaint`, just type `listmaint`. Here is what the top-level menu of `listmaint` looks like:

```
athena% listmaint

                                 List Menu
 1. (show)         Display information about a list.
 2. (add)          Create new List.
 3. (update)       Update characteristics of a list.
 4. (delete)       Delete a List.
 5. (query_remove) Interactively remove an item from all lists.
 6. (members)      Member Menu - Change/Show Members of a List..
 7. (list_info)    List Info Menu.
 8. (quotas)       Quota Menu.
 9. (help)         Print Help.
 t. (toggle)       Toggle logging on and off.
 q. (quit)         Quit.
 t. (toggle)       Toggle logging on and off.
 q. (quit)         Quit.
Command:
```

If you registered for your account before September, 1988, the first thing you will want to do is to claim rightful ownership of your personal list. To do this, you should first select *update* from the top-level menu. Once you do this, `listmaint` will ask you for the name of the list that you wish to update. Here, enter your username. Then `listmaint` will ask you several questions about the list. You can safely ignore most of these by just hitting return. The one you are interested is the question "`What Type of Administrator`". The current value will be show in square brackets. If you registered before September, 1988, the type will probably be *list*. You want the type to be *user* instead. You should type *user* at that prompt. You can then just hit return for the remaining questions until you are told that the list has been updated.

Another thing you may wish to do if you are using NFS is to add someone to your list. (If you don't know whether you're using NFS or AFS, you're almost certainly using AFS. If you have a standard Athena student account and you log in on public workstations or via dialup, you are using AFS.) If you do this, you can give that person extra access to your files. To learn how to do this, you can read the **File Protections** section of this guide.

To add someone to your list, select *members* from `listmaint`'s top-level menu. This will ask you for the name of a list and then put you in a submenu entitled `Change/Display membership of '`*list*`'`, where *list* is the list you specified. As you may expect, you will want to select *add* from this menu. You will be asked what type of member you wish to add. The answer to this question should be *user*. You will then be asked for the name of the user you wish to add. At this prompt, enter the user's username. You will be told whether the operation completed successfully.

Using `listmaint` is rather straight-forward. Once you get the hang of things, you will probably be able to figure out how to do mostly anything you want to.

You can also use `blanche` instead of listmaint. `blanche` can perform a number of the operations listmaint performs, and for those operations is much quicker. For a quick list of the functions `blanche` can perform, simply type *blanche*. For more detailed information, read the manpages.

Getting information through moira

In Unix, it is possible to `finger` a user to find out more information about him or her. To do this, just type `finger user@athena`. You can change this information about yourself with the `chfn` command. The use of the command is very straight forward. Just type `chfn`. You will then be told what your current information is and prompted for new information. The new information that you entered will be available to people who use `finger user@athena` within a day or so.

You can also find `finger user@mitdir` to get the "official" information about a person; this service is maintained by Distributed Computing and Network Services and any changes of information must go through the Registrar's Office (for students) or Personnel (for staff). You cannot change this finger information yourself.

What are you waiting for?

One last point. Because of the way *Moira* works, the changes you make won't take effect right away. If you add someone to your list, *Moira* will know right away, but the rest of the world won't know for between 15 minutes and a day or so. You may remember that your account wasn't ready until the day after you registered. In future versions of *Moira*, it may be possible to do this updating without delay.

But wait! There's more

There are more *Moira* programs than just `listmaint` and `chfn`. The program `moira` is the master *Moira* client. You could also have changed any of this information from the `moira` program, but it would have been slightly more complicated to do so.

To find more information on these other programs, refer to `moira` locker, which contains binaries and manual pages for the various moira clients. These programs are listed in the file README in */mit/moira*.

Acknowledgments and other randomness

The first edition of this document was produced and orchestrated by Jean Marie Diaz. Varying amounts of comments, criticism, text, and sympathy were provided by Simson L. Garfinkel, Henry Holtzman, John Kohl, Robert Krawitz, Henry Mensch, Ken Raeburn, Bill Sommerfeld, and Stan Zanarotti. Thanks, guys!

Second edition thanks to Rob French for RCSifying, tidying up, and generally letting Jean ignore this thing over IAP 1987, and to Ike Chuang, who valiantly volunteered to take over the *Guide* after Rob punted. (You haven't gotten away with anything, Ike, you'll get volunteered for it next time...).

The third edition encompassed revisions and additions reflecting Athena's migration from time-sharing to single-user workstations. John Kohl coordinated the conversion and edited the third edition. Ken Raeburn and Bill Sommerfeld (and other anonymous SIPB members and friends) helped edit early drafts. André DeHon, Mark Eichin, Jon Kamens, John Kohl, and Carol Smith contributed new sections.

The fourth edition, involving only minor changes, was revised to take into account Project Athena's version 6.0C software release. Jon Kamens edited the fourth edition. Andrew Greene and Emanuel ("Jay") Berkenbilt contributed new sections.

The fifth edition, which includes information relevant to Athena's version 6.3B software release and new sections on Xpix, AFS, and Andrew, was edited by Rob French (thus redeeming himself for punting on revision two). Many thanks to Jay Berkenbilt for writing the AFS section and revising some of the others, Seth Finkelstein and Barry Jaspan for writing the Xpix section, and Bill Cattey for writing the Andrew section.

Unix™ is a trademark of AT&T Bell Laboratories. They don't know that we're doing this, so don't tell them.

This document was created with GNU Emacs and LaTeX, and printed on the SIPB Hewlett-Packard 4si MX, using the Times Roman font family (hacked to use Courier-Bold for typewriter text). The source for the document is on the *sipb* filesystem in the directory `/mit/sipb/doc/guide`.