ProLine

- Installation
- User Tutorial
- Online Reference

Morgan Davis Group

Copyright

Sixth printing — November 1992 — U.S.A.

Printed on
recycled paper.

Contents

Personal Note

You are holding the latest manual for a software product that has consumed nearly every waking moment of my adult life—or so it seems. Almost ten years ago, I started writing a BBS for my brand new Apple IIe, because there were no bulletin board systems that supported my modem back then.

My goal was to provide the basic features of a BBS published by Southwestern Data Systems, known today as Roger Wagner Publishing. Their Apple DOS 3.3-based BBS, called *Online*, was written by Bill Blue. My Online-like BBS would run under Apple's new operating system, ProDOS, and was cleverly dubbed *ProLine*. It was a limitless project. There was always work to be done, always another feature to add.

It soon supported external Hayes-compatible modems and adopted a UNIX file system design, a solid platform for online information services. It grew so large that it wouldn't fit on a floppy disk. Soon it could exchange mail with itself and UNIX computers, in a limited capacity. Later, Internet mail and newsgroup processing was added. Today it barely fits on two 800K diskettes.

ProLine's software milestones mark personal ones for me. ProLine and I started out in junior college, transferred to San Diego State University, then shared time between various jobs. ProLine introduced me to many friends and acquaintances, some long forgotten, but many who are good friends today, like Bill Blue. It allowed me to start my own software business, and helped me buy my first home. It was the reason that I met Dawn, sysop of pro-simasd. We married a couple of years later and have two wonderful kids!

ProLine is more to me than just an item on our product list. It has permeated every aspect of my life for the last decade. If you have ever worried about investing in a product that might not be supported after you take it out of the box, rest assured that this is probably the most secure and valuable purchase you've ever made.

—Morgan Davis

Introduction

What Is ProLine?

Whether your world consists of classes or conference rooms, counties or continents, the ProLine bulletin board system brings people together. Now you can "link" the neighborhood with the rest of the world! With *networked* electronic mail, files, and bulletins, your Apple becomes a portal of information with the *Internet*, a global connection of computer networks.

The Internet is a super data highway allowing millions of people to participate in worldwide discussion groups. It even connects folks on commercial services, like America Online, AppleLink, BIX, CompuServe, Delphi, MCI Mail, and The Well, with users of networked bulletin board systems such as ProLine and FidoNet.

Unlike most systems that clutch onto one feature and ignore everything else, ProLine not only communicates with the Internet—an impressive feature by itself—but is perhaps the most complete BBS for any personal computer. It includes powerful local electronic mail, personal file management and storage for each user, automatic and unattended system maintenance, comprehensive online help, and incorporates the latest BBS technology available today. It is friendly *and* powerful, very easy to install, and is backed by the Morgan Davis Group with uncommonly responsive customer support.

About This Book

This manual is actually three books in one:

◊ Installation
◊ User Tutorial
◊ Online Reference

Part One, **Installation**, helps you prepare your computer system, and shows you how to install the software onto your drive. Installation takes about 30 minutes, depending on the speed of your computer system. After the ProLine software is installed, you configure it to work in the most optimum fashion with your hardware, then create your own account on the new system. Now you're ready to begin learning about ProLine.

Part Two, **User Tutorial**, takes you on a quick tour of ProLine, showing all of its major features, such as working with electronic mail, conferencing, and file transfers. The tutorial gives you an overview of ProLine's features. You'll want to explore ProLine in more detail.

Part Three, **Online Reference**, explains the hundreds of features that make up ProLine. Information in the Online Reference is reproduced from ProLine's online documentation. Every aspect of ProLine is described in the reference section in clear, concise language that helps you master ProLine.

Keys

Throughout this manual the following symbols are used to denote keys on your keyboard:

◀reset	Reset
option	Option or solid-apple
command	Command or open-apple
control	Control
esc	Escape
return	Return
delete	Delete
↑	Up arrow or control-K
↓	Down arrow or control-J
←	Left arrow or control-H
→	Right arrow or control-U
tab	Tab or control-I
shift	Shift

Hyphenated key references, such as command-esc, tell you to press and hold the first key while typing the second.

Part One:

Installation

Getting Started

This chapter shows you how to prepare your computer for running ProLine. You'll learn about cables, modem settings, and everything needed to allow ProLine to run on your system.

What You Should Know

In order for ProLine to work correctly with your computer, you may have to make adjustments to your hardware. You should be comfortable working with your computer's peripherals, installing interface cards, setting DIP switches, and if you have an Apple IIGS, making adjustments in the Control Panel.

You should be familiar with modems and telecommunications in general. Though not required, it is helpful if you have some experience as a user of a bulletin board system. Previous experience with a ProLine BBS is even more advantageous.

Hardware Supported

ProLine runs under ProDOS-8 on the Apple IIe and Apple IIGS, and is compatible with:

◊ Apple IIGS modem and printer ports
◊ Super Serial Card (and clones)
◊ Internal Hayes-compatible modem cards
◊ External modems
◊ Serial and Parallel printer interfaces
◊ AppleTalk networks

In addition to a modem and communications port, ProLine requires:

◊ A correctly wired modem cable
◊ A hard disk with at least 5 megabytes of free space
◊ A ProDOS-compatible clock card (or internal clock)

Internal Modem Cards

Internal modems mimic the combination of a Super Serial Card and a Hayes-compatible modem. ProLine should be treated as if just such a combination exists in the computer with an internal modem card. Read the sections describing the settings for the Super Serial Card and external modems.

Super Serial Card

To use ProLine with a Super Serial Card, the cable shown on the opposite page should be used with your modem. This wiring scheme provides hardware handshaking flow control (required by today's faster modems), data carrier detect, and support for DTR disconnections.

The special cable allows ProLine to detect the actual carrier signal on the phone line. Without it, ProLine could not know when the modem has disconnected with a remote modem. The diagram shows that DCD is tied to the CTS line. This is required by the Super Serial Card— it cannot communicate with the modem unless it believes that a carrier is present. The actual DCD signal from the modem is mapped to the DSR line at the serial port. ProLine reads this line for carrier status.

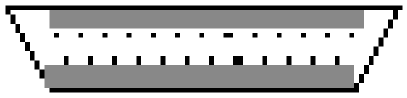
The Super Serial Card's jumper block must point to **Modem**, and the significant DIP switches should be set as follows:

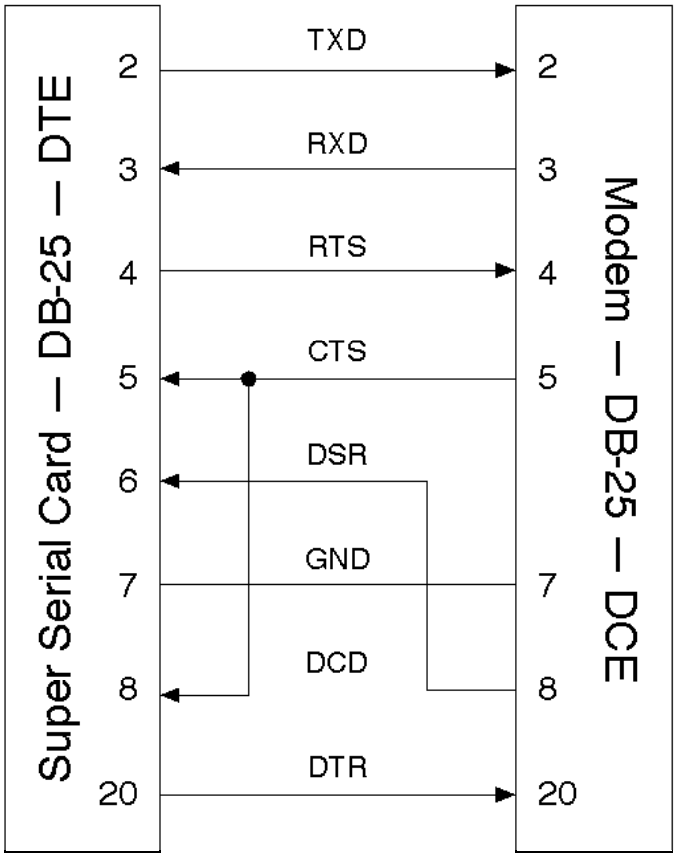
◊ Communications mode (not Printer mode)
◊ Interrupts are on
◊ RS-232C signals are on

You should set the default operating speed of the card to the highest speed that your modem supports.

NOTE: To use a Super Serial Card in an Apple IIGS, make sure to visit the Control Panel. Set the slot containing the serial card to ***Your Card****.*

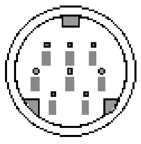
NOTE: The overhead in managing AppleTalk networking causes the Super Serial Card to lose data, especially at high speeds. Use the built-in Modem port for communications on the Apple IIGS. If you must communicate using the Super Serial Card, turn AppleTalk off.


DB-25 Connector

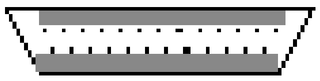


Apple IIGS Modem or Printer Port

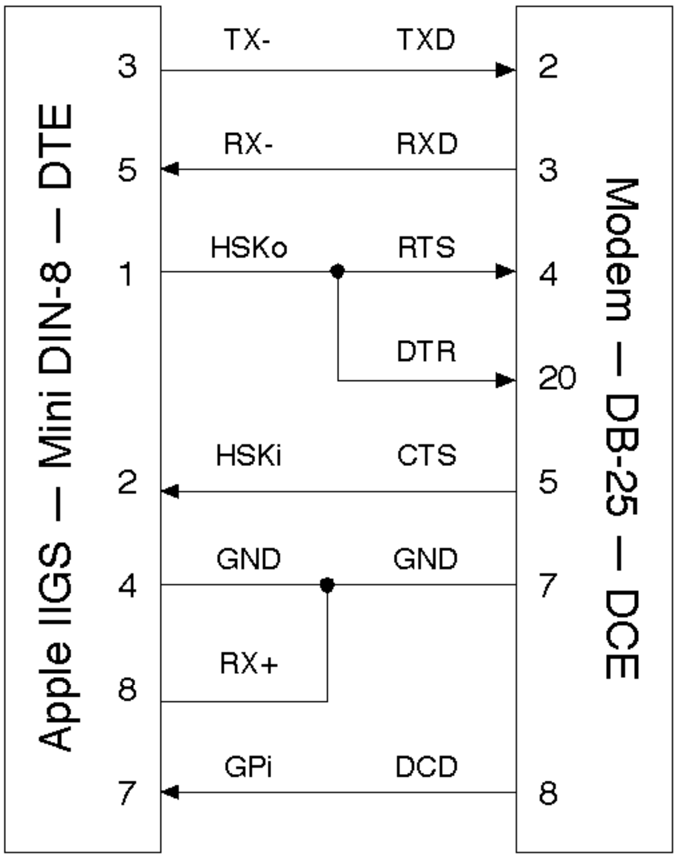
To use ProLine with the Apple IIGS built-in Modem or Printer ports, the following cable is required. This design is standard among high-end communications products for the Apple IIGS and Macintosh (which has a similar port design).



Mini DIN-8 Connector



DB-25 Connector



This cable provides true data carrier detection (DCD). It also supports hardware handshaking by connecting the Apple IIGS's two handshaking lines (HSKi and HSKo) to the modem's CTS and RTS lines. If hardware handshaking is not used, this wiring optionally supports DTR for disconnecting. The DTR wiring option is not required if the cable is to be used with only hardware handshaking.

WARNING: Since this cable allows the Apple IIGS's output handshake line to control two different modem signals, problems may occur if the modem is configured to respond to both RTS and DTR signals from the computer. With this cable, the modem must be configured to recognize only RTS or only DTR—not both at the same time. If hardware handshaking is used, the modem should be configured to ignore DTR.

The transmit-data and receive-data lines of the Apple IIGS serial interface conform to the EIA standard RS-422, which differs from the more commonly used RS-232-C standard. An RS-422 signal is less prone to noise and interference, and degrades less over distance than an RS-232-C signal. By grounding the positive side of each RS-422 receiver (RX+) and leaving the positive side of each transmitter (TX+) disconnected, as illustrated in the diagram, the cable essentially converts to EIA standard RS-423, and can be used to communicate with most RS-232-C devices over distances up to fifty feet.

Visit the Classic Desk Accessory menu by pressing command-control-esc. Open the Control Panel, and select Modem Port or Printer Port depending on the port your modem is connected to. Make sure the following items are set as indicated:

```
√ Device Connected: Modem
  DCD Handshake: No
√ DSR/DTR Handshake: Yes
```

ProLine can control the other settings.

*NOTE: If you do not use the recommended cable, as shown on the opposite page, set the **DSR/DTR Handshake** item to **No**.*

External Modems

For proper operation with ProLine, your external modem should be configured to these settings (if supported):

◊ Send verbal (not numeric) result codes
◊ Report the actual carrier status (support DCD)
◊ Enable command recognition

If you are using a hardware handshaking cable:

◊ Enable bidirectional RTS/CTS flow control
◊ Ignore Data Terminal Ready (DTR)—*Apple IIGS only*

Otherwise, if you are *not* using a hardware handshaking cable:

◊ Disable bidirectional RTS/CTS flow control
◊ Enable Data Terminal Ready (DTR) for disconnecting

Newer modems use terminal mode commands to set these options, to be saved in the modem's built-in memory. Older modems use DIP switches for these configurations. See your modem's manual for details.

High Speed Modems

ProLine provides support for high speed modems rated at 9600 bps or faster, or 2400 bps with data compression. For maximum performance, ProLine prefers that your computer and modem operate at the same speed all the time with a hardware handshaking cable. This is done by "locking the port" at the highest speed your computer and modem can handle. Although, your modem might be advertised as a 9600 bps modem, it might be able to communicate with your computer's serial port at 19,200 bps.

The Apple IIGS has a maximum port rate of 19,200 bps with standard software. The Apple Super Serial Card tops out at 9600 bps, but can be used at 19,200 bps on accellerated systems.

Custom Modem Control

As shipped, ProLine includes files containing information on nearly 50 modems. Called *modem capability files*, or *modemcaps*, they describe each modem's tolerances and features.

Each modemcap is preset to take full advantage of a modem's features. That is, the modem will operate at a fixed speed, at the highest rate supported, through the use of a hardware handshaking cable.

NOTE: Depending on your computer, serial interface, and cable, your system may not be able to accomodate the full abilities of your modem. For example, a stock Apple IIe may not be able to keep up with the data flow at 19,200 bps. You may have to make adjustments to your modem's modemcap file to provide compatible operation with your system.

ProLine comes with a modemcap file editor, called **mc**. **Mc** lets you change existing modemcap files as well as create new ones to the specifications required by your system, or to your own personal preferences.

ProLine Disks

ProLine comes on two 3.5 inch diskettes:

ProLine.1 ProLine.2

The **ProLine.1** disk contains the ProLine installation and configuration utilities. This is the disk you will startup with to install ProLine. Both disks contain the ProLine software. The contents of these disks are copied to your hard disk by the installation program.

You should make working copies of these diskettes and file the originals in a safe place.

Install ProLine

To install the ProLine software, follow these steps:

1. **Insert the ProLine.1 disk into your disk drive. Make sure the disk can be written to. *It must not be write protected.***

2. **Restart your computer, or use the Finder or a program selector to run the Startup program.**

   The ProLine Tools Main Menu appears.

3. **Choose the first item, *Read Important Notes*.**

   This displays important information about ProLine that you won't find in this manual. Read it now to learn about changes, additions, last minute notes, and other late-breaking news.

4. **Choose *Install/Upgrade ProLine.***

   The ProLine installation program appears, asking if you want to install a new system or upgrade an old one. Use the ← and → arrow keys to move the highlight bar to the **Install** item, then press return to select it.

5. **Select the location on your hard disk for ProLine.**

   The installation program asks you to specify the location on your hard disk where the ProLine software should be installed. This location, given as a ProDOS pathname, is known as the **ProLine System Directory**.

NOTE: You must decide if ProLine should be installed into your top level volume directory, referred to as the "root" directory (e.g. /A), or into a subdirectory (e.g. /A/PL). If you are dedicating your entire disk to your BBS, it is best to select the drive's root volume. If you plan to share your hard disk with lots of other applications and files, you might want to put ProLine into a subdirectory. Note, however, that placing ProLine in, or as close to the root volume as possible increases the speed of disk operations while running ProLine. Use short directory names, too.

   This manual assumes that you're installing ProLine into a subdirectory on a volume called "A". The directory name you've chosen is PL. You would enter:

```
ProLine System Directory: /a/pl   return
```

   If the PL directory does not exist, the installation program creates it for you, so you need not give the name of an existing directory on your drive.

6. **Install the first ProLine Backup disk.**

   The contents of the **ProLine.1** disk are copied into your ProLine System Directory first. This process takes about 15 minutes, depending on the speed of your computer.

7. **Is there another disk? (Y/N) Yes**

   After the contents of the first backup disk are copied, you're asked if there is another disk to restore. Remove ProLine.1 and replace it with ProLine.2, then press **Y** for Yes. Repeat this step for any additional backup disks that came with your ProLine software. When all the backup disks have been restored, press **N** for No.

8. **Insert the ProLine.1 disk.**

   The software installation process is complete. Insert the **ProLine.1** disk to return to the Main Menu. Now ProLine can be configured.

Configure ProLine

At the ProLine Tools Main Menu, select **Configure ProLine**. The configuration screen appears. Select each menu item and adjust it to suit your system. Some items offer you a list of choices which you can select by moving a highlighted bar with the arrow keys.

```
A. Modem:              Std.Hayes.Modem
B. Modem Slot:         2
C. Modem Speaker:      Always Off
```

The first three items specify information about your modem. Pressing **A** asks you to enter the name of a modem file (entering **?** displays a list of them). Enter the name of the file for your modem, or choose one of the Standard Hayes-compatible entries. Press **B** to select the slot for your modem (the Apple IIGS Modem Port is Slot 2). Press **C** to select the modem's speaker attribute.

```
D. Printer:            Printer
E. Printer Slot:            1
```

The next two items describe your printer. Even if you do not have a printer, specify a slot that would likely interface with one (the Apple IIGS Printer Port is Slot 1).

```
F. Site Name:          pro-test
G. Domain:
```

Press **F** to enter a unique name for your ProLine site, used for networked electronic mail addressing. Site names must be one word, begin with **pro-**, and cannot contain spaces, periods, or any punctuation characters other than the one dash (-). The Internet has reserved the **pro-** prefix to identify ProLine systems, and all ProLine sites must abide by this naming convention. Take a moment to think of something clever and professional. The dictionary includes a lengthy section of words that begin with **pro**.

WARNING: It is imperative that the **Domain** *item remain blank until your site has 1) linked with another site on the network, 2) registered a map entry, and 3) received a routing database. Assigning a domain to your site before completing these steps will keep mail from reaching your system. You'll learn about networking later.*

```
     H. Sysop's Login:    mdavis
     I. Sysop's Name:          Morgan Davis
```

Press **H** to enter the electronic mail name assigned to the sysop's account. Like site names, the sysop's login name is one word with no dashes, but can contain periods as the only punctuation allowed. Do *not* use "sysop". (ProLine automatically forwards any mail addressed to sysop to your account). Choose a name that resembles your real name. For example, if you are John Public, your login name might be johnp or jpublic. Press **I** to enter your full name.

```
     J. Time Zone:        PDT
```

Press **J** to enter a three-letter time zone abbreviation for your geographic area.

```
     K. Temp Files:       /ram5/tmp
     L. Spool Files:      /ram5/spool
```

These last two items give the locations where ProLine will store temporary files. System performance increases if a RAM disk is used, at the risk of losing data during a power outage.

Press **K** to enter the path to a directory in which temporary files, such as messages in composition, are stored. This directory should have at least 64K of free space (the Apple IIe /RAM volume is adequate). Press **L** to designate the path to a directory for ProLine's management of larger amounts of temporary data. This directory should have at least 1MB of free space available. If a RAM disk is not available, or not desirable, using "tmp" or "spool" (with no path prefix given) is suggested.

WARNING: These items require the path to a subdirectory. Giving only a volume name results in a ProLine startup failure, as ProDOS volumes impose a restriction on the number of files that may be stored in them, while subdirectories do not. Thus, it is valid to use /ram/tmp , but not /ram.

Press the esc key when you're satisfied with the configuration. Your changes are saved, and you are returned to the ProLine Tools Main Menu.

Running ProLine

At the ProLine Tools Main Menu, make sure your modem is connected and turned on, and select **Run ProLine**. The screen clears and ProLine begins its startup process. This involves loading communication modules, initializing the modem, and preparing for ProLine's operation.

If ProLine has successfully initialized your modem and is now displaying a **Waiting** prompt, your system is now *online*. Congratulations! You can go to the next chapter.

If your system encountered trouble during startup, here is a checklist of problems and solutions that may help you get it running . If you need further assistance, call for technical support.

"Requires Apple IIe or Newer"
- You're attempting to run ProLine on an Apple II or Apple II+. These computers are not supported by ProLine.

"Old (Buggy) BASIC.System"
- ProLine refuses to run under versions of BASIC.System that contain bugs that would hinder correct operation. Copy BASIC.System from the ProLine Tools disk over your old copy.

“Improperly Configured”

- The configuration file does not exist. ProLine must be configured first before it can run.
- The configuration file is damaged or empty. Restart the ProLine Tools disk and reconfigure.

“Can’t Create Directory”

- A directory given in the ProLine configuration could not be created (the path may include non-existent directories), or the volume is offline.
- A root volume was given in the configuration for the temporary or spool directories. ProLine requires that these directories be located in a subdirectory on a volume. Reconfigure.

Unexpected Error

- Call for customer assistance.

Fails Initialization, Freezes, or Crashes

- Incorrectly configured Control Panel.
- Incorrectly wired cable.
- Modem is not turned on or connected.
- Modem is not configured properly.
- Wrong slot selected in configuration.
- Wrong modemcap file selected in configuration.
- Modemcap speed is too fast for the computer’s port. If a high-speed modem is being used on an Apple IIe or with a Super Serial Card, press **I** to ignore the failed initialization and continue with the installation. Later, you’ll need to use ProLine’s Modemcap Editor, **mc**, to lower the modem’s highest speed setting, and then restart ProLine. **Mc** is described in the **Online Reference** part of this manual.

Logging In

Now that ProLine is installed and configured, you need to set up the first user account—yours—so that you can log in and begin exploring ProLine. This chapter shows you how to log into ProLine to finish setting up your system.

login

After a successful startup, ProLine runs the **login** program. This program is characterized by a **Waiting** prompt at the bottom of the screen:

```
Waiting < 01:23:45 >
```

Login patiently waits for various events to occur, such as:

◊ The modem's phone line rings
◊ The seconds on the clock become 00
◊ You activate commands from the keyboard

When the phone line rings, ProLine answers and attempts to connect with the caller. Once the connection is established, the caller is asked to enter a login name and password before entry is granted.

While waiting for an event, **login** checks a list of tasks to perform every minute. When a task's scheduled time arrives, **login** initiates its commands to be carried out by ProLine. It is this mechanism that gives ProLine its automatic maintenance abilities.

As time passes, **login** erases the screen, only to to display the time bouncing around the display. This built-in screen saver protects the display from phosphor burn, a problem common with computer systems that remain turned on for long periods of time.

Keyboard Events

While the time is patiently ticking away, you may press the following keys:

◊ esc exits to BASIC. (Use RUN to return to **login**).
◊ Space Bar toggles the screen saver on and off
◊ return lets you log into the system

In addition, you can enter one-key "macros" to invoke various tasks. Predefined macros are described in the **Online Reference** for the **login** program, if you're interested in more details. However, one macro key that you must learn about now is the exclamation point (**!**).

When you press **!** at the Waiting prompt, the screen shows:

```
Waiting < 01:23:45 > !■
```

At this point, **login** waits for you to enter a command, and then press the return key. To get the hang of it, enter the **df** command.

```
Waiting < 01:23:45 > !df   return
```

The **df** command displays the amount of free space on all your drives, and is just one of many online utilities that come with ProLine. When **df** is done, control returns to the **login** program.

Introduction To PLUSH

All interaction with ProLine is done by logging in to the system by giving an account name and a password so that ProLine knows who is using it. In order to log into ProLine, you need an account. Since there are no accounts yet, the only way to interact with ProLine is by issuing commands at the Waiting prompt. This allows you to create your account, as well as completing the final steps in setting up your ProLine system.

At the Waiting prompt, type **!** and enter the **plush** command:

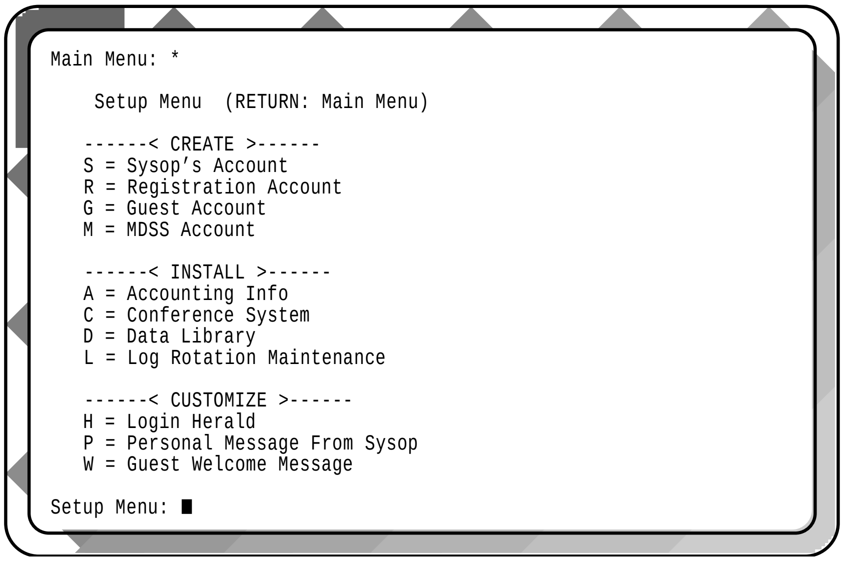
```
Waiting < 01:23:45 > !plush return
```

Plush is the ProLine Users Shell, a collection of menus that make it easy to navigate around ProLine.

NOTE: A "shell" is a program that provides the user with an interface that accepts commands, initiates them, and regains control when they're done. The underlying responsibility of all shells is the same, but their interfaces vary. ProLine comes with two kinds of shells: ***plush****, the ProLine Users Shell that offers menu-based command selection, and* ***csh****, the C-Shell, allowing the user to enter commands in a traditional "command line" interface. Both shells have advantages over the other. ProLine allows you to decide which one is right for you, but it initially selects* ***plush*** *as a user's shell.*

Plush first displays a Main Menu. From here, you can select ProLine's major features or open sub-menus to reveal additional features. Menu items are selected by typing in their corresponding letter. If an item opens another menu, you can always return to the previous menu by pressing return.

Right now, creating your own account is top priority. To do this, make sure you're at the Main Menu, and type an asterisk (*****), Shift-8 on your keyboard. This opens a hidden menu that is accessible only to you. After typing ***** your screen should show:


```
Main Menu: *

    Setup Menu  (RETURN: Main Menu)

   ------< CREATE >------
   S = Sysop's Account
   R = Registration Account
   G = Guest Account
   M = MDSS Account

   ------< INSTALL >------
   A = Accounting Info
   C = Conference System
   D = Data Library
   L = Log Rotation Maintenance

   ------< CUSTOMIZE >------
   H = Login Herald
   P = Personal Message From Sysop
   W = Guest Welcome Message

Setup Menu: ■
```

The Sysop Account

From the Setup Menu, press **S** to create an account for the system operator (sysop)—that's you. This menu item runs **adduser**, ProLine's Account Registration utility. **Adduser** asks for your first and last name, and prompts you with the name you entered when you configured ProLine.

```
Your first and last name    : Morgan Davis
```
return

Just press return. It then asks for your **login** identifier, and automatically inserts the name chosen during configuration.

```
Enter your LOGIN identifier: mdavis
```
return

Again, press return. Next, you are asked to select your secret password:

```
Enter your PASSWORD choice : ■
```

You must choose a password, at least four characters in length. The longer and more rich in letters, numbers, and symbols, the more secure your password is. As you enter your password, the keys you type are not displayed, just as they are not shown when you actually log into the system. You'll be asked to confirm your password by typing it a second time.

Finally, **adduser** asks that you enter your address and telephone number for accounting purposes. Once your account is created, you'll return to the Setup Menu.

*NOTE: To test out your new account now, press return to return to the Main Menu, then choose **B** (for Bye). Back at **login's** Waiting prompt, press return to log in using your new account and password. Enter your login identifier at the "login:" prompt, and your password at the "Password:" prompt. Eventually, **plush** displays its Main Menu, and from here, type ***** to return to the Setup Menu so you can continue setting up your system.*

The Register Account

When it comes to adding users to your system, a traditional BBS requires that users submit a request for accounts, which you then must verify and authorize before any kind of access is granted. ProLine takes a fresh approach to adding new users to your system. Rather than going through all that trouble for the 1% whose requests are denied, your ProLine can allow any user to sign on, sign up, and begin using ProLine with a personal account during the same phone call. Most commercial information services work this way.

SAGE ADVICE: Nearly all ProLine systems use the "open door" approach. This philosophy tells serious callers that you are fair and trusting, leaving a positive impression of your system and clientele. It attracts the kinds of users the sysop enjoys having around. It also discourages callers who are looking for trouble. Mischievous callers seek systems run by paranoid sysops, because they're a lot more fun to bother than professional systems run by benevolent operators. You can't "break into" an open system—there is no challenge. To be sure, the occasional twit will visit even the best of systems. As long as their attempts at upsetting your system are ignored, they will leave in search of a BBS with a less-experienced operator who reacts with the fire that fuels their folly.

This automatic registration feature is not activated unless you create a special **register** account, which you may add at your option. From the Setup Menu, select the **R** item to create the **register** account. Again, this invokes **adduser** which asks you to describe the account. Type in "Account Registration" as shown:

```
Description: Account Registration  return
```

Adduser does its best to select a login name for new accounts, but it isn't always the perfect choice. You will want to backspace and type in **register** instead of the name it suggests for the account:

```
Enter your LOGIN identifier: register  return
```

If you've decided to adopt the "open door" policy for new accounts, assign the **register** account with a password of **none**. Whenever **none** is selected for a password, **login** won't bother to ask the caller to enter any password at all.

```
Enter your PASSWORD choice : none return
```

If you would rather impose a little control over new accounts, you can give **register** a password that you share only with certain people. (As before, passwords are not shown when entered anywhere on ProLine).

Finally, **adduser** asks you to specify the "home" directory for the **register** account. This is not important for this kind of account, so simply enter *return* to accept what **adduser** suggests:

```
Home directory: /a/pl/  return
```

Your new **register** account will let a caller login (as "register") and then create a personal account, eliminating the administrative red tape that most bulletin board systems impose on the operator.

The Guest Account

You may want to create an account for guest callers, allowing them to visit your system without having to register for an account. A guest account has the lowest access rating, appropriate for first-time callers.

To create the **guest** account, type **G** at the Setup Menu. When **adduser** prompts for a first and last name, just leave it empty by pressing *return*.

```
Your first and last name    : return
```

Leaving the name item blank forces **login** to ask the guest caller to enter a name, city and state when logging in. Next, fill in the blanks as shown:

```
Enter your LOGIN identifier: guest return
Enter your PASSWORD choice : none return
```

This time, when asked to enter the **guest** account's home directory, type in **usr/guest** (note that there is no "e" in **usr**):

```
Home directory: /a/pl/usr/guest  return
```

The **guest** account is created.

The MDSS Account

If you plan to make use of ProLine's networking features (discussed later), you'll want to create the Mail Delivery Subsystem (MDSS) account. At the Setup Menu, type **M**.

Enter the following as shown:

```
Description: MDSS Account return
Enter your LOGIN identifier: mdss return
Enter your PASSWORD choice : whatever return
Home directory: return
```

When you enter the password, type in the word "whatever".

The **mdss** account is created.

Additional Set Up

The remaining items on the Setup Menu allow you to install and customize parts of ProLine. The Install items require the ProLine Tools Disk to be present.

◊ Install Accounting Info

You can adjust accounting information assigned to each new account added to the system (e.g. with the **register** account). The **adduser** program assumes certain default information, unless you specify otherwise. See the **adduser** description in **Administrator's** section of the **Online Reference**, notably the resource file settings.

◊ Install Conference System

The Conference System is where bulletins or messages are stored. Unlike traditional bulletin board systems that label these areas "bulletin boards" or "forums," on ProLine they are called conference areas because active participation is encouraged.

◊ Install Data Library

The Data Library is where users can upload and download computer programs.

◊ Install Log Rotation Maintenance

The Log Rotation Maintenance item installs seven directories on your system (one for each day of the week) to hold each day's log files. This allows you to maintain a week of log files on a rotating basis.

NOTE: ProLine automatically performs system maintenance at 3:20 A.M. and mails you a detailed report of all the tasks that took place, including a disk free space report, a list of callers who logged in, newly added accounts, and expired accounts. Additionally, if your ProLine is networked, the report includes network traffic statistics. With a number of different log files, it is left to you to decide how to dispose of them on a daily basis. Log rotation makes ProLine practically self-maintaining with no operator intervention required.

The Customize items invoke the ProLine text editor, allowing you to make changes to files on the system. By default, ProLine uses a line-oriented text editor, although it does include a full screen editor. You may want to visit the Preferences menu to select your text editor before doing any customizations. Refer to the **Online Reference** part of this manual for details on using **ed**, **edit**, and **vedit**, ProLine's text editors.
These are by no means the only areas that offer customization.

◊ Customize Login Herald

The **herald** is a file that the **login** program displays after each modem connection before the caller is prompted to enter a login and password code. This file usually contains a message that tells the caller the name of your ProLine sites, where it is located, communications information (e.g. “14.4Kbps V.32bis supported”), and any special login instructions (e.g. “Use GUEST to visit, or REGISTER to get an account”).

SAGE ADVICE: Lengthy herald files are discouraged, because they take up precious time for users and network sites calling long distance. Pictures, cute phrases, graphics, etc., can be placed in more appropriate areas such as ProLine’s “message of the day” file that callers see after logging in.

◊ Customize Personal Message From Sysop

Each new account is automatically mailed a greeting card from the system operator. This not only fosters a friendly atmosphere on ProLine, but shows your users that you care about them. Additionally, it allows new users to work with electronic mail during their first visit. Use this message to convey any system policies and guidelines as well.

◊ Customize Guest Welcome Message

When a guest caller signs on, the **welcome** file is displayed. Use this to introduce yourself and your ProLine system.

Where to Go from Here

Congratulations! Your ProLine system is officially installed and can begin serving you and the users of your BBS. Here are some steps you may wish to take from here:

◊ Skip to Part Two of this manual, the **User Tutorial**, and take the guided tour. Familiarize yourself with ProLine's features and functions from the perspective of a user. This allows you to learn ProLine so that you can offer assistance to your users.

◊ Read the next chapter, **Networking**, but only after you've become a veteran user. Networking is one of ProLine's hottest features, and can propel an ordinary BBS into an extraordinar-ily awesome system.

While exploring ProLine you often will refer to Part Three, the **Online Reference**. You will read a great number of entries in this section, but you need not immediately concern yourself with reading them all from start to finish. Some of the information in the **Online Reference** can easily overwhelm the new sysop who has yet to become a veteran user. Be sure to read the introduction to the **Online Reference** so you can use it properly.

*NOTE: The **C-Shell** has been mentioned in passing, even though it is the core of ProLine's power and abilities. You may not need to work with the C-Shell for a while (even though you may not realize that you already have). At some point, you will discover the side of ProLine that gives it true power and flexibility, and make the most of your investment. In the early days there was no ProLine Users Shell. Many ProLine operators who grew up with the C-Shell have been successful in adding impressive customizations to their systems. A healthy understanding of the C-Shell can be instrumental in designing extraordinary Plush menus.*

Up to this point, you have experienced a fraction of ProLine and its features. Since ProLine is a vast software package, you might become discouraged if you have not mastered it after a few days. Don’t be! Since ProLine is unlike any other BBS, you may find yourself exploring new and exciting areas of the system for months to come. Nobody really *masters* ProLine, not even its author! You advance in degrees of awareness and ProLine savvy. Consider all the applications that make up ProLine:

◊ An operating system
◊ An e-mail package
◊ A bulletin board system
◊ A networked mail and file server
◊ A massive collection of utilities
◊ A program selector
◊ A terminal program

To master ProLine in a couple of weeks is an unreasonable expectation, even if you are a UNIX wizard. There is too much to do and too much to explore. You have invested in a unique, engrossing, sweeping product, so give yourself ample time to experience ProLine at a leisurely pace. The rewards patiently await your discovery.

Programming

ProLine is a modular communications package patterned after the UNIX operating system. This architecture affords the highest degree of power and flexibility when it comes to adding new programs to your system. Being able to write software that works with ProLine adds unlimited power to your BBS.

ProLine gets its communications abilities from ModemWorks, a communications toolbox for BASIC programmers. ModemWorks takes care of much of the hard work in creating telecommunications software.

The applications that make up ProLine are Applesoft BASIC programs that follow rules for interacting in a shell environment. This allows user information, shell arguments, and configuration settings to be passed from the shell to its external programs.

To create your own ProLine applications, read the **plapp** entry in the **Software** section of the **Online Reference**. MD-BASIC and ModemWorks, both available from the Morgan Davis Group, are strongly recommended for any coding projects. They make ProLine development a breeze, which makes enhancing your system a pleasure.

Networking

ProLine is a wonderful bulletin board system, but it is also a sophisticated networked communications application.

This chapter introduces networking, giving pointers in getting your site connected with others. Before continuing, you should be familiar with the way the local mail system works. You might want to use an existing networked ProLine system to send out a few test letters, getting the hang of it, so that you'll know what to expect from your system.

The Networking Advantage

A BBS is a BBS is a BBS. But a *networked* BBS is something altogether different. Compare the value of a non-networked BBS and its modest gathering of local computer gurus to a networked BBS with great minds from all around the world. The advantage is obvious.

The network also is a great source of free information and software that can enhance your business, education, and hobbies. Many companies use ProLine for electronic technical support purposes. With a turnaround time 500% faster than the post office, online support is a speedy, cost-effective alternative when customers need help fast.

An advantage of being on the ProLine network is that you're in immediate touch with the Morgan Davis Group and friendly, experienced ProLine operators. They can answer your ProLine-related questions, no matter how technical. And if that isn't enough, many operators contribute new ProLine applications to the network. By networking, your BBS investment becomes increasingly valuable and rewarding.

Networking Glossary

The word *networking* is used so often that it is difficult to understand just what it might mean. On ProLine, networking describes the characteristics of two or more ProLine sites that automatically exchange electronic mail and files over the phone lines.

With help from features such as the cron task manager and powerful command shell environments, moving mail between ProLine sites is a smooth, slick process, seemingly more complicated than it really is.

Some networking terms you should know:

address — a mailing identifier that gives a user's login name and host system (e.g. user@site). Addresses sometimes include domains (see **domain**).

batch — a group of messages in one file to be unbatched into individual messages for posting in the conference system. (see **news**)

bounce — what a message does when it is addressed to an unknown recipient—it bounces back to the sender.

domain — a tag added to addresses to further describe a site's position in the network. Domains are to e-mail addresses as ZIP codes are to postal mail. (e.g. user@site.domain).

jobs, queues, traffic — electronic mail or files stored on a system, ready to be delivered.

map — a document that describes a site's connections and the costs associated with those connections.

MDSS — Mail Delivery SubSystem; the transfer scheme used to exchange mail between two connected sites.

news, newsgroup — A remote, electronic conference. A newsgroup is a continuous stream of related messages (batches), sent from site to site, containing information on a variety of subjects.

PMDSS — People-NET Mail Delivery SubSystem; a transfer scheme similar to MDSS, compatible between ProLine and People-NET (UNIX) hosts.

poll — to dial another system in order to exchange mail, usually for delivery after scanning.

scan — to scan the MDSS spool areas for pending traffic. If traffic exists for a site, then that site is polled.

UUCP — UNIX-to-UNIX CoPy; transfer scheme used by UNIX computers to exchange mail.

Common Questions

Here are some common questions asked about networking:

Q: Do I have to be part of the network?
No. Networking is an option (with many advantages). You can run your ProLine system just like any non-networked BBS. You are not expected to join the network, but you are always invited to do so.

Q: What does it cost to be on the network?
Nothing. There is no initiation fee, no monthly access charge, and no yearly renewal fee. All you pay for is standard phone usage.

Q: Which site must I talk to?
Site connections are not assigned. You can arrange to connect with any systems you desire. Contact the operators of the sites you wish to connect with after looking through the $/pub/proline/network.sites file.

Q: When should my site poll another?
That's up to you. ProLine gives you the flexibility of calling other sites whenever you want. Unlike other networked systems, ProLine systems can exchange mail at any time while servicing callers between poll times. Normally, your site would scan hourly, make local calls at that time, and only make long distance calls during the off-peak hours. See the **poll** entry in the **Online Reference** for more details on selective poll times.

Getting Connected

The first step in getting your system connected to another ProLine site is to read the **intro** manual entry in the NET subsection. This gives you an outline to follow. Read this now to get yourself up to speed on networking.

When your site becomes part of the ProLine network you must submit a map entry describing your system. Read the **map** manual entry for complete details in creating your map and how to submit it to the map coordinator.

You must determine when (and how often) you'll scan or poll your connected sites for mail. You need only to add an entry in your $/etc/crontab file to have these tasks executed at regular intervals.

Once you establish connections, request a subscription to the various ProLine newsgroups from one of your local connections. These newsgroups bring ProLine users and operators into world-wide discussion forums all about ProLine. They're like online magazines that keep you abreast of the latest ProLine news. See the **unbatch**, **rnews**, and **postnews** manual sections for details on setting up newsgroups.

*NOTE: To configure ProLine correctly, you left the **Domain** configuration item blank. When you finally receive your site's **paths** database file, you should fill in the domain entry. Most ProLine sites are governed under the Crash Time Sharing authority (a UNIX computer named **crash**), and use the **.cts.com** domain. Depending on your position on the network you might be better served by another domain authority.*

The Responsibility of Privilege

Becoming part of the ProLine network is a privilege, not a right. Along with it comes an enormous responsibility. Once networked, people rely on your site's presence and cooperation to keep mail flowing smoothy and reliably. A site that is susceptible to hardware trouble, or is offline for lengthy amounts of time, should not be on the network.

You must also insure that only professional and helpful activity emanates from your site and its users. You will have to police your users if they abuse the network (e.g. subscribing to newsgroups on their own without your permission). Operators will not hesitate to disconnect your site if it cannot comply with the standards and behavior that is expected of ProLine systems.

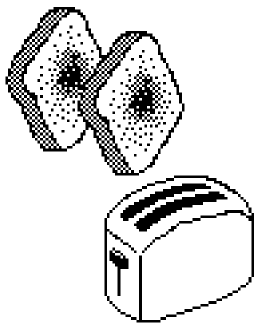
SAGE ADVICE: Users are guests of the net. Sysops are the stewards of the net — they care for it and maintain it. Nobody owns the net. For more on netiquette*, see the **tutorial** manual entry in the NET subsection.*

Part Two: User Tutorial

How To Make Toast...

You don’t want to spend hours reading the instructions before you can get the toaster to make toast. It takes only a minute to figure it out, even if the instructions consist of just a few terse steps:

1. Insert bread
2. Push button down
3. Wait for toast



Simple, right? That depends on how you describe “making toast.”

A toaster is a complex kitchen appliance when you consider the physics involved in heating bread with a machine. To fully understand every aspect of the process, the instructions would have to explain the principles of electricity (what makes it go), convection and variable capacitance (how it heats the bread), mechanical timing (how long it heats the bread), tension and resistance (what makes the toast pop up), and the thermochemical changes that turn a limp slice of bread into a crunchy piece of toast.

Each one of those items would require volumes of technical writing to describe them. We shouldn’t have to be scientists just to make a snack. All we want is toast and a little butter.

This is a tutorial about using ProLine, a bulletin board system (BBS) that you connect to using your computer's modem. Like a toaster, ProLine offers basic features that should be simple to figure out. On a BBS, you want to do things like:

◊ Exchange electronic mail

◊ Participate in discussion areas

◊ Get some new software

◊ Hangup, and make some toast

The tutorial walks you through the steps quickly so that you can start using these basic features right away. It won't bog you down with technical explanations of the options that abound on ProLine. You can explore those things on your own because, like a toaster, it would take a lot of reading to learn every aspect of ProLine.

The detailed technical information is provided, if you're inclined to study it, by ProLine's online documentation system. Once you have the basics down, working with ProLine's more advanced features is easy.

Right now, we just want to get online and start having fun . . .

Getting Started

This chapter shows you how to sign onto your local ProLine system, get your own account, and get to ProLine's Main Menu. From there, you'll be able to begin using ProLine's features, as described in the chapters that follow.

If you're using the computer that actually runs the ProLine BBS software (you're not calling in from a remote computer with a modem), ignore references to connections and modems, as they won't apply to you.

What You Should Know

The first thing you need to know is the phone number and operating speed of the ProLine system you're calling. You should already know how to use your terminal program. Fill in the blanks below for your own reference.

The ProLine system I call is:

The phone number is:

Connecting

Dial the ProLine system. When you connect, you'll see something like looks like this:

```
ProLine [pro-host]  Anytown USA

login: ■
```

Get Your Account

The ProLine host identifies itself, and is asking you to **log in**.

If you already have an account on the host computer, just sign on now using your login name and password. Then go to the next section.

Before you can use ProLine, you need to register for an account. Most ProLine systems use a special login name, **register**, that you enter to sign up for your own account. (Some systems may use different names like **new** or **newuser**). Enter the name your ProLine host specifies for obtaining an account.

```
login: register   return
```

This activates the ProLine Account Registration questionaire. Follow the directions carefully, filling in your name and address as instructed, and soon you'll have your own personal account. After your account is created, you are asked to activate it by logging in.

```
ProLine [pro-host]  Anytown USA

login: mylogin   return
Password: mypassword   return
```

NOTE: For security purposes, your password is not displayed when you type it.

Depending on the ProLine you're signing onto, you may be presented with all sorts of interesting things: random words of wisdom, news headlines, and so on. These are supplied by the operator of the system you're calling, and vary from system to system. All systems will tell you the date and time when you lasted called and if you have any mail waiting.

```
You have mail.
```

Main Menu

Eventually, ProLine displays its Main Menu.

```
   Main Menu

   C = Conference System
   E = Electronic Mail...
   F = File Library

   H = Help Desk...
   I = Information Desk...

   G = Game Room...
   M = Maintenance...
   P = Preferences...
   U = Utilities...

   B = Bye
   X = Expert Command Shell

Main Menu: ■
```

Again, since system operators can customize and enhance their systems, what you might see on your screen may differ from what is presented here. You may have to adjust yourself to the differences and follow along as best as you can.

Select Your Preferences

Before getting into the main areas of ProLine, such as Electronic Mail and Conferencing, ProLine should know about your computer's communications abilities. This allows ProLine to make the best use of your terminal program's features, such as screen emulation, to enhance your sessions.

Select the Preferences Menu item by pressing the **P** key.

This displays the Preferences menu:

```
Main Menu: P

   _Preferences_  (RETURN: Main Menu)

   E = Environment Settings
   P = Password Change

Preferences: ■
```

Let’s make adjustments to the communications environment, so press **E**. You get a new menu:

```
          ProLine Environment Editor

Settings for kepler (Johannes Kepler):

   1. Cancel Key:    Control-C
   2. Tabs Are:      Preserved
   3. Nulls:         0
   4. Screen Height: 24 lines
   5. Screen Width:  80 columns
   6. —More— Paging: Yes
   7. Terminal:      tty
   8. Menu Mode:          On
   9. Hot-Key Mode:  On
  10. Text Editor:   edit

Number to change (Q to quit): ■
```

The settings that are initially important are your Cancel Key, —More— Paging, Terminal, and Text Editor. To change an item, just enter its number and follow the on-screen instructions.

Cancel Key. Your cancel key is Control-C, but you can change it if you like (to something like Escape). When you press your cancel key, this tells ProLine to stop what it is doing and return to a point where it can accept new commands. It is great to have. If you *really* screw up and get yourself into a mess, press your cancel key.

—More— Paging. Each time the screen fills, ProLine lets you know that there is more, and waits for you to press a key so it can continue. If you’re using a slow modem and find these prompts annoying, you can turn this feature off permanently. Or, you can leave it on and temporarily disable it when you don't need it by pressing the minus key (**-**) at the **—More—** prompt. Most areas on ProLine allow you to turn it on again by pressing the plus key (**+**).

Terminal. By now, you've discovered that calling a BBS with a modem leaves much to be desired. A BBS like ProLine can't control the whizzy graphics capabilities of your computer to give you dialog boxes and pull down menus. You're stuck with a fairly droll display. Fortunately, if your terminal program supports it, you can jazz things up a little by enabling terminal emulation. Popular terminals supported by modern software are: ANSI, VT-100, VT-102, and VT-52. You can't go wrong with ANSI emulation, if your terminal program supports it (most do). This lets you take advantage of all of ProLine's screen emulation features. (See your terminal program's manual for more details on this esoteric subject).

Text Editor. Second to reading, you do a lot of writing while using ProLine. Whether you're composing mail or formulating a reply in your favorite discussion group, you will call upon one of ProLine's text editors to make corrections to your prose. Initially, ProLine selects a rather simple, but arcane line-oriented text editor (**edit**) which works for everyone on ProLine, regardless of Terminal Emulation. However, ProLine supports a full-screen text editor (**vedit**) that works much like a word processor. If you've selected adequate terminal emulation, you will find **vedit** to be far more enjoyable than **edit**. (Unless you're a UNIX nut, you won't want to use **ed**. If you are a UNIX nut, you don't need this tutorial!).

Once you've set things up as you like, type **Q** to quit back to the Preferences Menu. ProLine records your new environment settings for this and future sessions. You can always adjust your preferences again if needed.

Now, press *return* to return to the Main Menu. We're going to check Electronic Mail next . . .

Electronic Mail

Of all the areas on ProLine that you really need to know how to use effectively, electronic mail is your lifeline to assistance if you encounter difficulties. As long as you know how to send and read mail, you can keep in touch with the system operator or a knowledgeable contact who will assist you.

You Have Mail

If you recall, when you signed on, ProLine said you had mail. Let's go read it now.

At the Main Menu, press **E** to go to the Electronic Mail Menu.

```
   Electronic Mail  (RETURN: Main Menu)

   R = Read Your Mail
   S = Send Mail

   W = Write to the Sysop

   I = Internet Mail Guide
   M = Membership Directory...
   N = Network Directory

E-Mail: ■
```

Since you have mail, press **R** to read it. ProLine faithfully opens your mailbox and displays a summary of the letters waiting.

```
 _________________
|  ====       [] |_
|                | |
|                | |
|       Mail     | |
|________________| |
  |________________|

You have 1 message (768 bytes)

Msg #   Size  Date    From      Subject
    1    768  Jun 26  mdavis    Welcome!

mail> ■
```

This summary gives you an idea of what is in your mailbox. It shows the number of letters you have, their sizes, when they were written, who sent them, and what they're about.

The easiest way to read new mail in your mailbox is to press **N** for **Next**. This displays the next new message, or in this case, the first message in your mailbox. Press **N** now.

```
Message 1 of 1:

From mdavis Sun Jun 27 12:34:56 1992
Date: Sun, Jun 27 92 12:24:56 PDT
From: mdavis (Morgan Davis)
To: kepler
Subject: Welcome to ProLine!

Greetings!  I want to personally welcome you to
ProLine, the foremost BBS for enlightened souls
such as yourself.  If you should ever need
assistance, feel free to write to me.  I want
you to get the most from ProLine.

I'd like to know more about you, your
interests, and what you would like to get out
of this system.  Please reply now.

1 of 1: mail> ■
```

Well, you heard the man! Press **R** to **Reply**. ProLine asks:

```
1 of 1: mail> Reply to ■
```

Since your mailbox can hold more than one letter at a time, ProLine wants you to tell it which letter you want to reply to. With only one letter in your mailbox this may seem silly, but there are a number of responses you may give at this point which you will want to learn about later. For now, just press return to reply to the current letter (the one you just read):

```
1 of 1: mail> Reply to Current

To: mdavis■
```

Now, ProLine asks you to confirm the recipient of your reply, which it has automatically entered for you. The cursor is placed just after the name, allowing you to change it or add additional names. For now, just press return.

```
Subject: Re: Welcome to ProLine■
```

ProLine puts the cursor at the end of the new subject so you can backspace and change it. For now, just press return to accept it.

```
To: mdavis
Subject: Re: Welcome to ProLine

Enter your message.  Put a "." on a new line
when done.

:■
```

Now you're in message input mode where you can enter a reply. Take a moment to let the sysop know more about you. While entering your reply, ProLine does word wrapping, like a word processor, when you reach the right side of the screen. You needn't press return unless you want to start a new paragraph.

```
:I was born in Germany, 1571.  I have a
:strong interest in astronomy and
mathematics...
```

When done, press return to put the cursor at the beginning of a new line, then enter a single period (.) followed by return. This tells ProLine you're done entering text.

```
:...looking forward to reading sci.space
:and sci.astro here. return
:. return

1 of 1: mail>send> ■
```

Notice how the prompt has changed to show that you're in the process of composing a message for delivery. To send your message, press **S**.

```
1 of 1: mail>send> Sending to mdavis

1 of 1: mail> ■
```

When your reply has been sent, you return to the **mail>** prompt. Now mark the current letter for deletion by typing **D** and return:

```
1 of 1: mail> Delete Current

1* of 1: mail> ■
```

An asterisk (*) will appear next to messages you've marked for deletion. They're not really deleted until you quit, so if you change your mind you can undelete by typing **U**. To leave the mail program, type **Q**. You're asked to confirm your deletions:

```
1* of 1: mail> Quit -- delete all messages? Yes
```

Congratulations! You just read and replied to your first electronic letter. You'll find that the steps you've taken are typical of most of your electronic-mail sessions: read, reply, delete, quit.

When you return to the Electronic Mail menu, press return to go back to the Main Menu.

Conferencing

The Conference System is where system members gather to discuss a variety of topics in public or private forums. In other words, this is the area where you read and post messages for all to see.

Bulletin boards, like the one at the post office or drug store, are where people stand in silence to read something somebody posted with a thumb tack. In a Conference System, members engage in active conversations, making conferencing a social event. You get the feeling that people are participating right before your eyes with a lot of back-and-forth banter. That's why we call it a Conference System, not a Bulletin Board.

Getting Started

From the Main Menu, you enter the Conference System by typing **C**.

```
 __________
|    o-    |_
| --- - -- | |
| - --- -  | |   ProLine
| -- -- -- | |   Conference
|__________| |   System
   |_________|

Joined to 0 Conferences -- No new messages.

cs> ■
```

Upon entering the Conference System, you're shown a summary of new messages in the areas that you've joined. (Individual systems can automatically join you to particular conferences, so your display may look different than above).

A conference is dedicated to one particular subject and is further organized into topics. For example, a conference on **music** might include the following topics: **concerts**, **discs**, **reviews**, and **midi**. When you join a new conference, ProLine automatically joins you to each topic in that conference. You can *unjoin* yourself from individual topics, however.

The Conference System is so easy to use that the only key you may ever need to remember is return. This is because pressing return joins you to the next conference with new messages, and pressing it again displays the next unread message. Keep pressing return for each new message until there aren't any more left. Then press **Q** to **Quit**. It's that easy.

Joining A Conference

The first step in using the Conference System is to join the conferences that interest you. To get a listing of the available conferences, type **L** for **List** at the **cs>** prompt.

```
cs> List Conferences

apple2       Apple II series conference
chatter      Conference for sundry chit chat
ibm.pc       IBM PC (and clones) conference
macintosh    Macintosh series conference
proline      All about ProLine

cs> ■
```

To join a conference, press **J** for **Join**, then enter the name of the conference.

```
cs> Join chatter return

Welcome to the "chatter" Conference!
Joining chatter/misc, 100 new messages

cs>read> ■
```

If the conference has only one topic, ProLine takes you right to it. If there are more than one topics, you get to select the one you want to start reading first.

Reading Messages

Once you've joined a conference/topic, your prompt changes to show that you're now in the process of reading messages. Start off by pressing return to show the first new message. After each message is shown, press return again to go onto the next.

To skip over any messages, use the **Skip** command by pressing **S**:

```
cs>read> Skip to ■
```

The Conference System needs to be told what message you want to see. To skip over all messages, type **L** for **Last**:

```
cs>read> Skip to Last
```

The Conference System says:

```
LAST MESSAGE in conference/topic
cs>read> ■
```

Pressing return at this point takes you to the next topic with new messages. If there are no more new messages, you return to the **cs>** prompt.

Adding or Replying

ProLine lets you contribute to a discussion at any time. If you want to add a completely new thought to the topic, use the **Add** command by pressing **A**. If you just read a message that requires a reply, press **R**. When you **Reply**, the Conference System asks you which message you want to reply to (just like in electronic mail):

```
cs>read> Reply to ■
```

Usually, you'll tell it that you want to reply to the current message, the one you just read, by typing return (or **C** for **Current**).

When you Add or Reply, you are asked to enter the *Subject* of your contribution. Type in a brief description that summarizes what your message is about.

```
cs>read> Add message
Subject: How to make toast in zero-G  return
Msg #6502: enter text, end with "." alone.

:■
```

As with a pad of paper, you can write your message and edit it before you add it to the topic. When done entering text, type a period on a new line and press return. This takes you to the cs>read>add> prompt. To save your message, type **S** for **Save**.

```
cs>read>add> Saving 6502

cs>read> ■
```

Conferencing In A Nutshell

You now know the basics of conferencing. Your sessions will typically involve the following steps:

◊ Use **List** to list the available conferences
◊ Use **Join** to join those that interest you
◊ Press return after each new message until there are no more
◊ **Reply** to messages that invite your commentary
◊ **Quit** back to the Main Menu

File Library

In the File Library, you can check out software programs, but you won't lose your library card if you don't return them. Modelled after the school library you're familiar with, the ProLine File Library offers a unique approach for organizing computer files, making it easy to find the software you want. Take all your items to the Front Desk and check them out (download them) all at once. Simple.

The File Library is friendly enough to figure out without having to read lots of instructions, but it includes comprehensive online help at its Information Desk. You'll want to visit the Information Desk and go on the Guided Tour. Check out the other Information Desk items as well (to bone up on the Dewey Decimal System?).

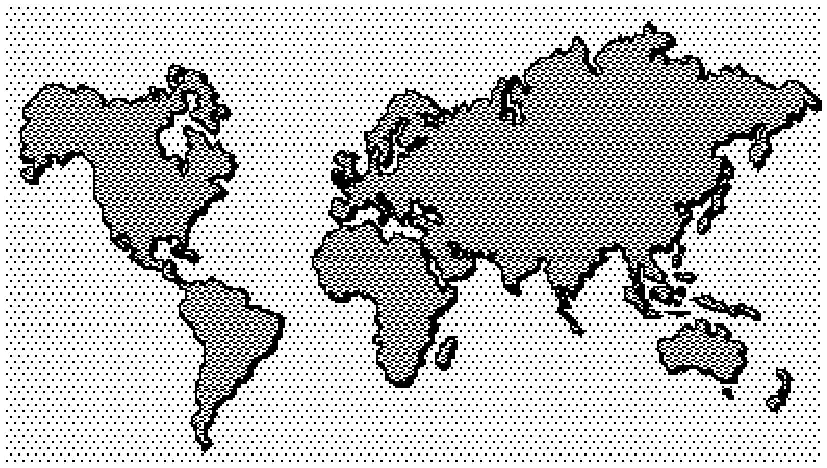
You'll want to familiarize yourself with your terminal program's upload and download protocols before exploring the library. See your program's manual for details.

Reach Out . . .

You know that ProLine lets you communicate with users on the system you call through electronic mail and the Conference System, but did you know that you can also exchange mail and conferencing messages with users on computer systems around the world?

The Global Village Is Here

This exciting feature is made possible by ProLine's *networking* capabilities. ProLine can link with other ProLine computers, as well as with a global network of computers known collectively as *The Internet*. This network includes all kinds of computer systems: personal computers, workstations, mini and mainframe computers, and even supercomputers! These computers are run by hobbyists, universities, corporate industries, commercial information services, and the government. They all cooperate to move messages and files around the globe faster than traditional paper mail—an impressive feat.



Networking has been around for some time, but is just now becoming known to the general public. ProLine has been networking since 1985, but only today are the major information services getting into the act, such as America Online, AppleLink, BYTE Information Exchange, CompuServe, MCI Mail and others. With so many computers plugging into the Internet, it is now possible to communicate with anyone who has a modem, no matter where they are, even in space! (In 1991, astronauts aboard the space shuttle used a Macintosh portable to send a message to AppleLink).

Sending Net Mail

Each computer on the network has a unique name, used as an address. To send mail to a friend on another computer in the network, you give your friend's name and the name of the computer he or she visits. For example, to send a letter to **jsbach** who is on **pro-musette**, you address your message as:

```
To: jsbach@pro-musette
```

This reads, "send mail to user jsbach at site pro-musette". Easy! (All ProLine site names begin with a **pro-** prefix).

Domains

The Internet is organized in much the same way as our postal system, with regional "main" post offices and smaller neighborhood stations. When you send a letter from your home to someone in another state, the letter first stops at your neighborhood station. Then it gets driven downtown to the main post office where it is scheduled for delivery by plane to another main post office. The cycle reverses, and your letter is delivered from a neighborhood station to your friend's house.

Imagine the chaos if letter carriers picked up mail and hand-carried it directly to its destination. The cost would be enormous and it would take forever. Fortunately, the post office uses a system of "hubs" to efficiently deliver mail.

The Internet uses an even more efficient system of routing electronic mail through major hubs. This is based on a system of "domains" where sections of the Internet are responsible for handling mail within their domain. As an example, all ProLine systems know how to route mail to any other ProLine system, but computers outside of the ProLine network have no knowledge of each ProLine site. If each site on the Internet had to know about every other site, it would impose a tremendous burden on computers—analogous to a letter carrier having to hand-carry a letter to its destination.

In the Internet, sites pass letters through the domain system. In the postal system, a mail carrier takes your letter back to the neighborhood station. In the Internet, the computer that services a domain passes your letter to another domain authority. In the postal system, a station delivers your letter to other stations. Eventually, Internet mail gets to a destination domain, which knows about the destination site. Postal mail gets to a destination station which knows about the destination street. You get the idea.

If site names are like street addresses, then how do you denote a domain? Domains, like ZIP codes which govern a region of postal addresses, govern a number of sites. You attach domain names to address names like this:

```
To: user@site.domain
```

This tells your site to deliver a message to the computer that governs **domain**. It is that computer's job to get the message to **site**.

Sometimes you may need to include more than one domain in an address:

```
To: user@site.domain.domain.domain
```

There are a number of address examples in the next section.

Islands in the Net

You can send mail to people on computer services that support Internet mail. Below is a list of some of them with examples showing how you address them.

Service	Description	Sample Address
America Online	America Online, Inc.	• *user*@aol.com
AppleLink	Apple Computer, Inc.	• *user*@applelink.apple.com
ATTMail	AT&T, Inc.	• *user*@attmail.com
Bitnet	Int'l academic network	• *user*%site.bitnet@*gateway*
BIX	General Videotex Corp.	• *user*@dcibix.das.net
BMUG	Berkeley Mac User's Group	• *First.Last*@bmug.fidonet.org
CompuServe	CompuServe Info Services	• *71234.567*@compuserve.com
Connect	Professional Info Network	• *NAME*@dcjcon.das.net
EasyNet	Digital Equipment Corp.	• *user@host*.enet.dec.com
		• *user%host*.enet@decwrl.dec.com
		• *First.Last@ABC*.MTS.DEC.COM
Envoy	Canadian mail service	• attmail.com!mhs!envoy!*userid*
Fidonet	PC-based BBS network	• *First.Last*@p*4*.f*3*.n*2*.z*1*.fidonet.org
GEnie	General Electric Corp.	•
GeoNet	GeoNet Mailbox Systems	• *user*:geo4@map.das.net
MCI Mail	MCI mail service	• *First.Last*@mcimail.com
		• *1234567*@mcimail.com
MFENet	Magnetic Fusion Energy Net	• *user%mfenode*.mfenet@nmfecc.arpa
NASAMail	NASA internal e-mail	• *user*@nasamail.nasa.gov
PeaceNet	Non-profit mail service	• *user*%cdp@arisia.xerox.com
SINet	Schlumberger Int'l Network	• *user@node*.SINet.SLB.COM
		• *user%node@node1*.SINet.SLB.COM
SPAN	Space Physics Analysis Net	• *user@host*.span.NASA.gov
		• *user%host*.span@ames.arc.nasa.gov
SprintMail	Sprint's mail service	• /C=*country*/ADMD=*system*/ O=*organization*/PN=*First_Last*/ DD.ID=*userid*/@Sprint.COM
TheNet	Texas Higher Education Net	• *user%host*.decnet@utadnx.cc. utexas.edu

Internet Guidelines

Using the Internet entails responsible behavior. This can't be stressed enough. System operators deal with discourteous users and abuse of the Internet on a daily basis. They have little patience for such trouble, even if due to honest mistakes. Much has been written about what you should and should not do on the Internet, but here is an essential list of the things you should know and understand:

Keep messages short. The Internet is free for you to access, but somebody is paying to bring it to you. System operators foot the bill for their long distance connections. The less data you have to send, the less the operators have to pay. Make each word count, because each word *does* count.

Use short context helpers. When you reply to a message, it is often helpful to *briefly* paraphrase or "quote" passages of the message to bring your reply into context. Many people don't do this, and readers have trouble following their replies. Others include too much information, only to follow it with a one-line response. Quote only relevant passages, and paraphrase if they're too long.

Respond only when appropriate. In the Conference System, read all the messages before responding to a question for which you have an answer. Somebody may have already posted an answer. Duplicate responses waste network space and transfer time.

Avoid online wars. If somebody points out that you misspelled a word, ignore it. If somebody says that your computer isn't as good as theirs, ignore it. Massive "flame wars" have started because people got involved in pointless arguments. Save time and money by not contributing to such worthless drivel.

Sarcasm is not in the ASCII set. Computer screens can't convey irony or deep sarcasm. A humorous or sarcastic statement should be followed with a **:-)**, which looks like a smiley-face turned sideways. It tells others that you're "just kidding."

You're communicating with people, not computers. It is all too easy to blow your top online and regret it later. The computer shields us from people and the damage we can do to them through emotionally heated replies. Always remember that there is a warm-blooded human on the other side of your monitor.

Graduation

Congratulations! You now know how to work with the major areas on ProLine. Here are a few final notes before you begin your long and productive relationship with ProLine.

Common Commands

Most areas of ProLine recognize a common set of commands so that you don't have to shift gears each time you go from one part of the system to another. Here are some command keys you can enter just about anywhere on ProLine:

Control-S	Stops and starts the display.
Cancel Key	Takes you to the most recent command prompt.
?	Displays helpful information.
+	Turns on --More-- paging.
-	Turns off --More-- paging.
Q	Quits the section of ProLine you're in, returning you to the previous command level.

At the --More-- prompt:

Q	Acts like you pressed your Cancel Key.
-	Turns off --More-- paging.

Any other key resumes output until the screen fills again.

More Details

This tutorial quickly introduced you to the major features of ProLine. Each area, such as electronic mail or the conference system, includes many commands. A few of them are basic and simple—which you learned about here. There are other commands that you will want to explore that make ProLine easier to use than most bulletin board systems. ProLine can do the hard work that other systems make you do manually.

To find out more about ProLine's major features, consult the Online Reference manuals. Simply visit the Help Desk from the Main Menu after you sign on. The Help Desk includes detailed descriptions of the Conference System, Electronic Mail, the File Library, using the Network, and ProLine's command line shell. You can also get a listing of the hundreds of other topics in the Online Reference, then get information on individual subjects.

Road Map

As you'll discover, there is more to explore on ProLine than what has been presented in this tutorial. Here are some other features you may want to read up on in the Online Reference manuals:

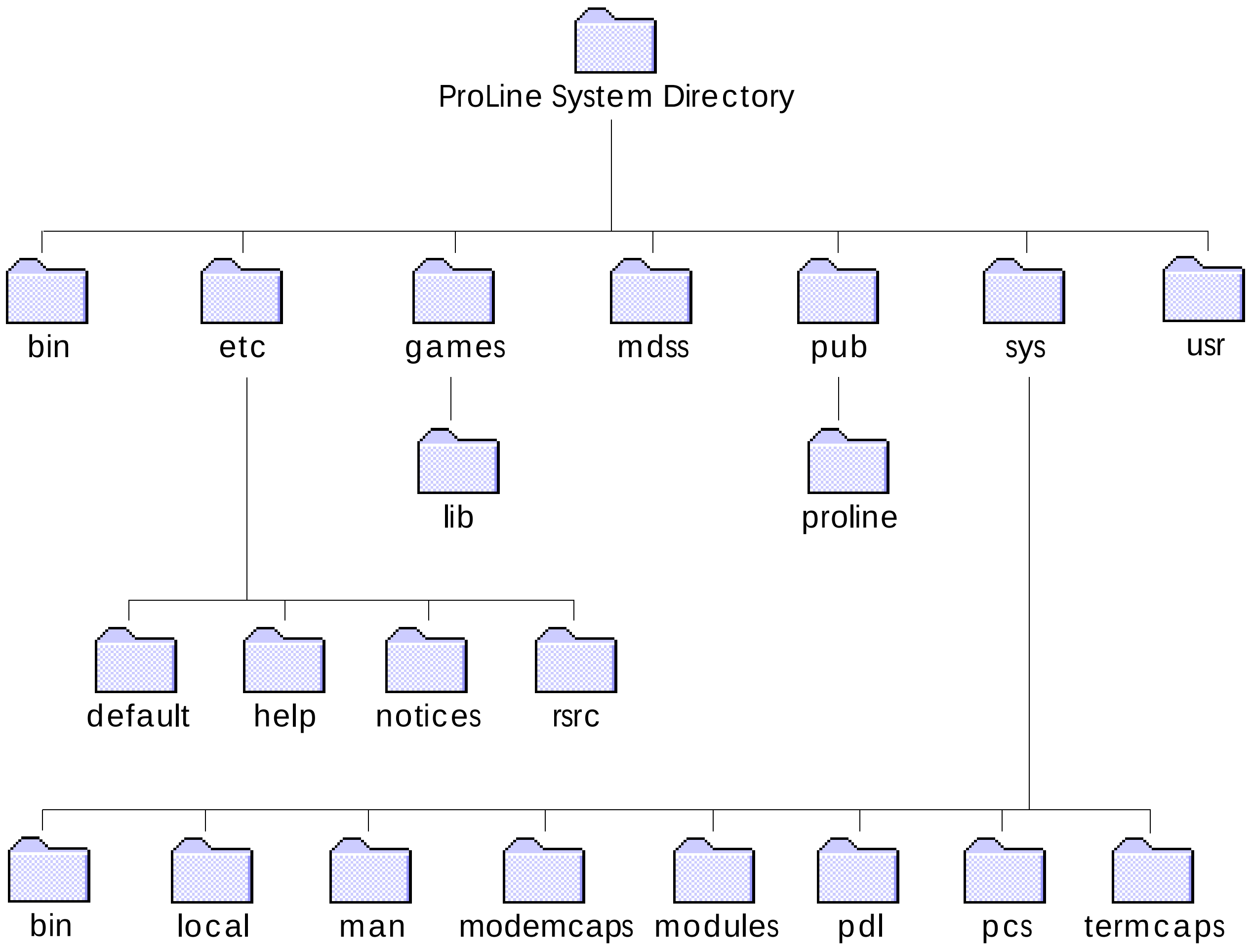
◊ Your own personal directory in which you can store files and letters (see **csh**, **login, sx, rx, sz,** and **rz**)

◊ The ability to share and exchange files with other users' personal directories (see **cat, csh**, **cp, dstat**)

◊ The ability to customize and program your own sessions on ProLine (see **csh** and **scripts**)

◊ Network file server—allows you to obtain files from remote systems (see **server**)

Once you start getting into these advanced areas, you'll need a map of the system with which to navigate. See the opposite page.



Legend

bin	Programs and utilities
etc	Miscellaneous files
etc/default	Default files
etc/help	Help files
etc/notices	Copyright and other notices
etc/rsrc	Resource files
games	Online games
games/lib	Score files, etc.
mdss	Offsite spool directories
pub	Public files directory
pub/proline	Public ProLine files
sys	System directories
sys/bin	Programs and utilities
sys/local	Local system additions
sys/man	Online manuals
sys/modemcaps	Modem capability files
sys/modules	Communications modules
sys/pcs	ProLine Conference System files
sys/pdl	ProLine Data Library files
sys/termcaps	Terminal capability files
usr	User directories

Part Three: Online Reference

Introduction

ProLine's extensive online documentation, reproduced on the pages following, cover a wide range of subjects. To organize the online documentation, the following sections are provided:

Section	Description
ADM	System Administration
C	General Commands
CT	Typesetting Commands
CP	Programming Commands
F	File Formats
G	Online Games
HW	Hardware Dependent
LOCAL	Local Utilities
M	Miscellaneous
NET	Networking
S	Software Subroutines and Libraries

Each section consists of *entries*, and each entry describes a particular part of ProLine. To locate an entry, you need to know the section it falls under, and then look it up. Sections and entries are arranged alphabetically, like a dictionary, for easy access. Sections are shown in parenthesis after the entry's name.

Entries follow a specific format, illustrated here in this typical sample entry:

SAMPLE(M)   ProLine Reference Manual   SAMPLE(M)

Name

sample - This one-line summary describes the entry

Syntax

sample syntax shows how to use the command

Description

Begins the main part of the entry describing all the details.

Files

Lists any files related to the entry.

See Also

Lists entries related to this one (suggested reading)

Revised 27 Feb 1992   1   Printed 7 Sep 1992

Syntax

Special characters are used in the **Syntax** part of the entry. Items enclosed in brackets, **[** like this **]**, denote optional parameters. Don't include the brackets, of course.

Items separated by the pipe (|) character denote choices. For example, ***AA* | *BB***, means you include either **AA** or **BB** as an argument.

Elipses (**. . .**) means "more of the same." For example, ***file* . . .**, tells you that one or more file arguments may be given.

Administration

addconf	(ADM) - Add a conference
addtopic	(ADM) - Add a topic to the conference system
adduser	(ADM) - Adds new accounts
boot	(ADM) - Shuts down and reboots
cron	(ADM) - Execute scheduled tasks
cronsw	(ADM) - Switch crontab files
cs.maint	(ADM) - Conference system maintenance
dl.maint	(ADM) - Data librarian maintenance
eduser	(ADM) - Edit a user's account
flip	(ADM) - Flag inactive persons for removal
gid	(ADM) - Group ID utility
maint	(ADM) - System maintenance
mc	(ADM) - Modemcap editor
mount	(ADM) - Mount disk devices
notice	(ADM) - Display a notice
rc	(ADM) - Command script for automatic restart
rmconf	(ADM) - Remove a conference
rmtopic	(ADM) - Remove a conference topic
rmuser	(ADM) - Remove users
startup	(ADM) - System configuration and startup
synctime	(ADM) - Synchronize clock to an atomic time reference
tc	(ADM) - Termcap editor
unmount	(ADM) - Unmount disk devices
updates	(ADM) - How to install software updates

Commands

add	(C) - Append files
alias, unalias	(C) - Alias or unalias command names
banner	(C) - Print message using big letters
basic	(C) - BASIC interpreter
calendar	(C) - Personal calendar
calls	(C) - Report system usage
cat	(C) - Concatenate and print
cd	(C) - Change working directory
chat	(C) - Chat mode
cp	(C) - Copy a file
cs	(C) - Conference System
csh	(C) - Command shell
csmod	(C) - Conference System moderator's utility

ctime	(C) - Connect time and accounting info
df	(C) - Disk free space
dl	(C) - Data Librarian
dstat	(C) - Directory access status
du	(C) - Summarize disk usage
echo	(C) - Echo text arguments
ed	(C) - Text editor
edit	(C) - Text editor
err	(C) - Describe error codes
find	(C) - Find files
grep	(C) - Find a pattern in a file
help	(C) - Get help
hist	(C) - System history
if	(C) - Conditional command execution
import, export	(C) - Text file conversion
it	(C) - InteleTerm Pro terminal program
itc	(C) - InteleTerm Pro script compiler
log	(C) - Display log files
login	(C) - Sign on
logout	(C) - Terminate connection
lpr	(C) - Send files to the printer
ls	(C) - List directory contents
mail	(C) - Electronic mail
mcinews	(C) - Poll MCI Mail for business news headlines
mkdir	(C) - Make a directory
more	(C) - Browse text files
mv	(C) - Move a file
news	(C) - Manage system news bulletins
od	(C) - Output dump in hex
par	(C) - ProLine archive utility
passwd	(C) - Change login password
pc	(C) - Programmable calculator
plush	(C) - ProLine Users Shell
pwd	(C) - Print working directory name
resume	(C) - Resume file generator
rm, rmdir	(C) - Remove files or directories
rot13	(C) - Rot13 file conversion utility
rx	(C) - Receive files using XMODEM or YMODEM
rz	(C) - Receive files using ZMODEM

safecom	(C) - File encryption utility
sendmail	(C) - Send mail over the internet
server	(C) - Internet file server
set, unset	(C) - Set or unset shell variables
setenv	(C) - Set environment
setfile	(C) - Set file attributes
sleep	(C) - Suspend execution for an interval
sm	(C) - Send modem command
sort	(C) - Sort a file
source	(C) - Read shell commands from a file
split	(C) - Split a file into pieces
stty	(C) - Set terminal options
sweep	(C) - File utility
sx	(C) - Send files using XMODEM or YMODEM
sz	(C) - Send files using ZMODEM
tail	(C) - Deliver the last part of a file
trim	(C) - Trim mailbox headers
tset	(C) - Set terminal emulation
unpar	(C) - ProLine archive unpacker
uuencode, uudecode	(C) - Encode/decode a binary file for mailing
vedit	(C) - Visual text editor
wc	(C) - Word count
who	(C) - Who is on the system

Typesetting

makebook	(CT) - Create a file containing the entire manual
man	(CT) - Prints pages in the manual
manps	(CT) - Print PostScript-formatted pages in the manual
whatis	(CT) - Summarizes commands

Formats

cs	(F) - Conference System file formats
manuals	(F) - Format of manuals
plush	(F) - Menu format and programming

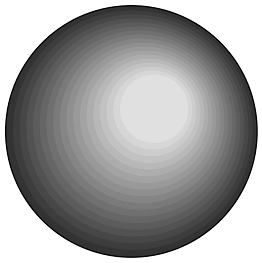
Games

af	(G) - Add a fortune to the fortunes file
fortune	(G) - Print a random, hopefully interesting, adage
today	(G) - Events on this day throughout history

Miscellaneous	scripts	(M) - All about shell scripts
Networking	aliases	(NET) - Description of mail alias files
	batch	(NET) - Process mailboxes into newsgroups
	domains	(NET) - All about Internet domains
	intro	(NET) - Introduction to networking
	map	(NET) - How to create a map entry file
	mdss	(NET) - Mail Delivery SubSystem
	mdssclean	(NET) - Mail Delivery Sub-System cleanup
	mknull	(NET) - Make remote letters addressed to null
	mksig	(NET) - Make signatures
	netauth	(NET) - Authorize network mail access
	path	(NET) - Show the mail route to a site
	poll	(NET) - Poll a site
	postnews	(NET) - Post newsgroups to local news areas
	rcp	(NET) - Remote file copy
	rmnull	(NET) - Remove null files
	rnews	(NET) - Distribute newsgroup bundles
	scan	(NET) - Scan network mailboxes
	tutorial	(NET) - Networked e-mail tutorial
	unbatch	(NET) - Convert rnews batches into newsgroup bundles
	uutraf	(NET) - Network traffic report
Software	plapp	(S) - How to create ProLine applications

ProLine

System Administration



Name

addconf - Add a conference

Syntax

addconf

Description

Addconf allows the Super User to add a conference (with topics) to the conference system. The program walks you through the creation of the conference, from choosing its name and a one-line description, and specifying each topic.

Adding a conference involves these steps:

1. Choose a name. Conference names must be legal ProDOS file names, therefore must begin with a letter, contain letters, digits, or periods, and be 15 characters or less in length. Use periods for spacing.

2. Set access. Conferences can be open or closed. An open conference can be joined by anyone. If you create a closed conference, members can only be joined by a Super User or the moderator of the conference.

3. You must select a moderator for the conference. The moderator has the ability to assume temporary Super User access while joined to the conference. This allows them to remove messages and post in READ-ONLY topics. Enter the moderator's login name.

4. Enter a description of the conference. Each conference includes a one-line description for display in the conference listing file. If you enter nothing, the conference will not be shown in the listing.

5. *Addconf* asks you to enter the name of each topic, up to six topics total. When done entering topic names, press RETURN. Like conference names, topics must also be legal ProDOS file names.

6. For each topic you create, you're asked if the topic is Read-Only. Read-only means that members cannot add new messages. You're also asked if the topic is networked. A networked topic is one in which local postings are sent to an e-mail address. If the topic is networked, you'll be asked for the e-mail address, the corresponding newsgroup name (if applicable), and whether or not signature files should be attached to postings.

Note: Postings to Usenet groups should be sent to the rnews mailbox at the local participating Unix site (e.g., rnews@cts.com).

Files

$/sys/pcs/cs.list - description file of conferences,
$/etc/rsrc/cs.rsrc - contains the path to the cs area.

See Also

addtopic(ADM), cs(C), cs.maint(ADM), csmod(C), rmconf(ADM), rmtopic(ADM)

Name

addtopic - Add a topic to the conference system

Syntax

addtopic [*conf/topic*] [-rs] [-p *address*] [-n *newsgroup*]

Description

Addtopic adds a topic to an existing conference. Up to six topics may be added to a conference. *Addtopic* can be used interactively or non-interactively as needed.

When invoked without arguments, *addtopic* prompts you to enter various information to create the new topic. It first asks you to select the conference, then asks for a name for the new topic. Topic names are legal file names: 15 characters or less, start with a letter, and include only letters, numbers and periods. Topics can be created with read-only access if desired.

A topic can also be assigned networking information. A networked topic is one in which posted articles are mailed to a recipient for further processing and/or distribution. Articles posted to a networked topic do not show up immediately. In addition to a mail address, a networked topic can also include the formal name of a newsgroup under which it is to be distributed. Finally, *adduser* allows you to specify if users' signature files should be attached to articles posted in the topic.

To use *addtopic* in a non-interactive mode, include the following arguments (all but the first are optional):

conf/topic	Specifies the existing conference (*conf*) and the new *topic* to be created.
-p address	Assigns a posting *address* (thus defining a networked topic). This usually references the rnews address at a participating Unix site, or the address of a mail alias or moderator on another site.
-n newsgroup	Assigns a *newsgroup* name, required for topics that contain USENET news articles. The -p argument is required in order to direct posted articles to a USENET site's rnews account, or the mail alias for a ProLine-originated newsgroup.
-r	Enables read-only access.
-s	Enables attachment of signature files to posted articles.

Examples

```
addtopic
addtopic programming/basic
addtopic proline/announce -r
addtopic info/mailing -s -p info-moderator@site.domain
addtopic org/mdg -p pro-org-mdg@pro-sol -n pro.org.mdg
addtopic apple2/group -p rnews@cts.com -n comp.sys.apple2
```

See Also

addconf(ADM), cs(F), csmod(C), rmtopic(ADM)

Name

adduser, mkuser - Adds new accounts

Syntax

adduser [*options*] [*interpreter*]

Description

Adduser and *mkuser* work together to add new accounts to the system. Both require Super User (root) status to run. Without any arguments, *adduser* creates a *staff* level account, and uses *csh* as the command interpreter. The command interpreter, usually a shell of some sort, is the program that is run after the user logs in successfully. If an *interpreter* argument is given, it is used in place of *csh*.

The procedure for adding a new account involves choosing an account login name and password. Login names are 15 characters or less, begin with a letter, and can contain only letters or digits. Passwords can be up to 12 characters, are not case sensitive, and can contain any printable characters. If a password of ‘‘none’’ is chosen, the account is not asked for a password when logging in.

Options can be offered in the command line, instructing *adduser* to create different types of accounts. Options for specifying group type are:

-r *root* account. This level offers full privileges and total access to the system. Only one account on the system should have root status with access to an interpreter such as the C-Shell.

-s *staff* account. This level is for most users on the system. Staff users can be completely mobile around the system yet cannot view or access any sensitive material.

-m *mail* (network) account. This is for use by network computers in order to exchange e-mail, news, etc. This account does not have a user area.

-g *guest* account. This is the same as a staff account, except the guest must enter a name and location if no name is given to this account. Guest accounts cannot modify their Conference System information file.

These options specify other account attributes:

-a bypass accounting. This assigns a user-ID of 0 to the account. No user area is created, and no user information is requested. *Adduser* will only ask for a description of the account being created.

-l do not copy the default login and cshrc scripts to the user’s home directory. This is desired when assigning a command interpreter other than *csh*, as only *csh* executes these scripts.

-u do not prompt for user information, useful when creating a guest account. Usually, *adduser* asks for a user’s full name, address, and phone number.

-n no network access. This prevents a user from sending mail to offsite addresses in the network.

-i do not create a signature file for the user.

-w do not mail the welcome file to the user's mailbox.

Examples

```
adduser -gun plush
```

Create a guest account, no user info, no network mail access, use plush (the ProLine User's SHell) as the interpreter.

```
adduser -m mdss
```

Create a network mail login entry. Use mdss as the interpreter.

```
 adduser -rau "adduser -s csh"
```

Create a login entry which runs *adduser* as the interpreter with options for setting up a staff account. This allows a caller to create an account without any administrator intervention. Although this example creates a root level account (*adduser* can only be run from a root process), the person using it causes *adduser* to create a personal staff level account.

If an account's name or description is left blank, a login attempt using that account name will be prompted for a name and location. This is mainly used for guest accounts.

Creating a User Account

A typical user account is created in the following manner:

1. *Adduser* asks the user to fill in a number of information fields, then creates an entry in *$/etc/passwd* (see below for entry description). An account record is created for the user in *$/etc/adm*, a database containing statistics for each user of the system. An entry containing information about the newly created account is appended to the adduser.log file.

2. *Adduser* then transfers control to *mkuser*, a shell script that can further complete the installation of user accounts. *Mkuser* creates the user's home directory. The default login and cshrc scripts are copied here for the user. A *signature* file may also be created.

3. A directory with the user's login name is created in *$/adm*. This area is used by the system for storing the user's data files.

4. A welcome file is sent to the user’s mailbox. This file contains a personal message from the administrator.

The _Mkuser_ Script

Since *adduser* can only create user accounts when it is invoked with super user status, that allows *mkuser* to perform similar super user functions. It is important to understand that *mkuser* is run under the parent process that invoked *adduser* -- IT IS NOT BEING RUN AS IF IT WERE OWNED BY THE NEW USER BEING CREATED.

This means the user--AT NO TIME--should ever get interactive access to any part of the system that might lead to a security problem. Furthermore, invoking commands intended to effect the user being created is futile because the effective user is most likely to be root, the system administrator, or an account registration user with super user status, NOT the intended user.

Mkuser is provided to allow customized file control while an account is being created. In addition, the distribution version of *mkuser* lives in $/etc and should not be modified. A copy of *mkuser* can be placed into the effective user’s ‘‘~/bin’’ directory or into the system’s local ‘‘bin’’ directory (e.g., $/sys/local/bin) such that it takes precedence over the distribution version.

Password File Entry

Adduser alphabetically inserts a new account entry into the password file. Password entries have this format:

```
jdoe:ASji8aQd3pxYg:142:1:John Doe:usr/jdoe:csh
```

This corresponds to:

Login : Password : UID : GID : Name : Home : Interpreter

The *user-id* (UID) is a unique number identifying the user. The *group-id* (GID) is one of the following:

0 - Root-level account
1 - Staff-level account
2 - Network mail account
3 - Guest-level account

The user’s home directory and interpreter pathnames do not start with the system’s startup prefix ($/). This is added by *login* so that any name change of the startup prefix avoids a complete update of the password file. A path to the interpreter is not needed unless it does not reside in any of the standard directories for programs.

Resource File

Adduser reads a resource file (*$/etc/rsrc/adduser.rsrc*) containing information on account settings when adding a new user. The resource file consists of five lines of information:

Minutes Per Month	The total minutes per month the user can spend on the system each month. If the user overspends his allowance for the current month, access is denied until the beginning of the next month. Time allowances are reset at the beginning of each month. Default 1200 (40 minutes per day average). If set to zero, the time allowance is unlimited.
Inactive Days	The number of days that may pass without logging in before the user is flagged for removal due to inactivity. If set to zero, the inactivity removal feature is disabled. Default 60 days.
Cents Per Minute	The cost for access time in pennies per minute. A value of 1.5 would denote a 1.5 cent per minute access charge, or 90 cents per hour. Default 0 cents per minute (no access charge).
Max Accounts	The number of accounts that can be active at any one time. Default 200 users.
Leniency Level	If 0, *adduser* is specific in accepting input entered by the user. For example, it refuses to continue until the user enters exactly ten digits for the phone number. If 1, it is lenient about input, useful for systems that receive calls from users who live outside of the United States. Default 0 (no leniency)

If the resource file is not present, the default values are used.

Files

$/etc/adm - accounting database,
$spool/logs/adduser.log - log of new user accounts,
$/etc/passwd - password file,
$/etc/mkuser - shell script for customized user account creation,
$/etc/help/adduser - introductory help file,
$/etc/default/mail - welcome letter,
$/etc/default/login - default login script,
$/etc/default/cshrc - default cshrc script,
$/etc/rsrc/adduser.rsrc - resource file,
$/adm/* - user's system data directory,
$/usr/* - user's home directory.

See Also

startup(ADM), eduser(ADM), login(C), passwd(C), rmuser(ADM)

Name

boot - Shuts down and reboots

Syntax

boot [*option*] [*app* | *slot*]

Description

Using *boot* is the preferred method for taking the system offline. Without any arguments, *boot* shuts down the system and reboots.

Boot recognizes these options:

-b	Enters BASIC immediate mode.
-h	Does not hangup if there is a connection.
-q	Quits BASIC and returns to the operating system.
-r	Restarts the system by running the $/ProLine program.

If *boot* is given a numeric argument from 1 to 7, it attempts to boot the computer from the device in that slot.

If the argument is a filename of an executable file, that file is launched.

See Also

startup(ADM)

Name

cron - Execute scheduled tasks

Description

Cron is a task that operates once every minute from within the *login* program while the system is idle (waiting for callers or for a console instruction). It executes commands at specified dates and times according to the schedule in the file $/etc/crontab.

Sample crontab file:

```
Day  ## Month  HH MM Command
*    *  *      *  09 csh echo "Scanning..."; scan
*    *  *      03 20 csh maint -r
*    *  *      06 30 lpr /hd0/spool/lpr
*    *  *      18 30 lpr /hd0/spool/lpr
Sun  *  *      00 00 boot
*    *  Jul    03 30 csh doublemaint
*     1 Jan    00 00 banner Happy New Year!
```

Each line in crontab, good for one time event, contains six fields of information. The arguments in each field must be correctly placed under the field header, the first line in the file.

The six fields, from left to right, are the day of the week (string), date of the month (integer), month of the year (string), the hour (integer), the minute (integer), and a command with any necessary arguments. Unless a path is explicitly given, commands are assumed to reside in $/bin.

All hour and minute arguments must fill the entire width of their associated field (i.e., ‘‘03’’ for ‘‘3’’). The date of the month is padded with spaces (see the above example for January 1). If an argument starts with an asterisk (*), that entire argument is treated as a wildcard (it will count as a matched time pattern).

In the above example, the system will ‘‘scan’’ for outgoing mail every 9 minutes on the hour. It will run the ‘‘maint’’ C-Shell script every morning at 3:20. Any queued printer jobs in /hd0/spool/lpr will be printed twice a day at 6:30am and 6:30pm. Every Sunday at midnight, the system will reboot itself. During the entire month of July, the double maintenance C-Shell script is executed at 3:30 in the morning. And on January 1, the system will wish you a Happy New Year.

It is best to put high-frequency tasks before other tasks, as shown in the above example. Also note that if you need to perform two or more tasks at or about the same time, you can have crontab reference a single shell script that will perform all the appropriate duties.

Note that internal shell commands, such as ‘‘echo’’, and shell scripts must be called using the name of the shell first, as in ‘‘csh echo Hello World’’. This is because *cron* assumes the first argument in the command line is a program in the $/bin directory.

Files

$/etc/crontab - task table.

Note

Since this isn’t a true multitasking computer, tasks in crontab will only run when the system is not busy doing something else. Understand that some jobs might not get executed. If one task follows a previous task within a span of a minute or two, they should be invoked from a single shell script, one after the other.

See Also

sleep(C)

Name

cronsw - Switch crontab files

Syntax

cronsw *tag*

Description

Cronsw, a shell script, allows the system administrator to switch crontab files. The *tag* argument specifies the suffix of a ‘‘cron’’ file to switch to. For example, if the *tag* is ‘‘day’’, the $/etc/cron.day file is copied to $/etc/crontab file, and becomes the current task schedule.

Files

$/etc/cron.*tag* - files accepted as input by cronsw,
$/etc/crontab - output file of cronsw, used by system to schedule tasks.

See Also

cron(ADM), scripts(M)

Author

Dean Fick (dean@pro-electric)

Name

cs.maint - Conference system maintenance

Syntax

cs.maint [conference]

Description

Cs.maint is a full screen, menu driven program that aids in the maintenance of the conference system. It allows the system administrator to change the names of conferences and topics, moderators, posting addresses for networked conferences, and the read-only status of topics. If a conference name is specified, *cs.maint* acts as if the find option was used to select that conference.

Note

When a conference is renamed, it is treated as if you removed the old conference, and created a new one, hence the users must re-subscribe to the conference.

See Also

cs(C), csmod(C)

Author

Daniel Davidson. Bugs or suggestions should be sent to danield@pro-grouch.

Name

dl.maint - Data librarian maintenance

Syntax

dl.maint

Description

(Throughout this manual entry, *dl* denotes the prefix to the Data Library's location as stored in $/sys/pdl/dl.dir).

This program performs five major functions in relation to the data librarian program:

- maintenance of the card catalog (*dl*/lib/dl.data), the program subdirectory (*dl*/lib/prog/*), and the program description subdirectory (*dl*/lib/desc/*).
- maintenance of the library shelves (*dl*/lib/dl.vars). A maximum of 16 shelves are allowed.
- maintenance of index files (*dl*/lib/link/*) that contain the names of all programs in the Data Library that are related to one another (see subheading on LINK FILES).
- configuration of certain ‘‘preference’’ settings, such as the assignment of a librarian other than the system administrator, requiring a minimum baud rate for Data Library access, guest caller access limits, the maximum number of downloads allowed per session, upload filesize limit, and freespace (volume overhead) limit.
- moving ‘‘archived’’ programs to offline storage, freeing up disk space.

Options supported for file-oriented maintenance are:

Edit	Edit the record displayed on the screen.
Delete	Delete the record, program and description from the Data Library.
Find	Find the record matching the program title entered. Using a ‘‘*’’ will instruct *dl.maint* to find all titles that are ‘‘on hold’’.
Next	Display the next record in the Data Library.
Previous	Display the previous record in the Data Library.
Quit	Exit the maintenance subroutine and return to the maintenance menu.

Link Files

When the user downloads files placed on reserve the program will open the *dl*/lib/link/* file and search for any program/data files that are ‘‘related’’ to the selections placed on reserve. If related files are found the user will be given the option of including these ‘‘linked’’ files in the download.

For example, let's assume that there is a program called EDITOR in the library that has associated documentation (EDITOR.DOCS) and will require the ShrinkIt program to ''decompress'' the program file. Since the user will need copies of EDITOR.DOCS and ShrinkIt to use the EDITOR program the librarian should add them to the *dl*/lib/link/* file using option #3 (Edit Link List) from the maintenance menu.

Notes

All information related to a record in the Data Library can be changed except for the program filename stored in the *dl*/lib/prog subdirectory. The program filename must remain unchanged in order for the link/support programs to work correctly!

All programs should be thoroughly tested before their status is changed from ''on hold'' to ''available'' so that pirated or non-working programs aren't downloaded by users.

Even though a program has been ''archived'' (moved to offline storage) the card catalog entry will remain online. The entry must be deleted from the card catalog if you wish to remove all references to the program.

Important

If you intend to access *dl.maint* via dialup (remote) your telecommunications program must support some type of terminal emulation mode.

Files

dl/lib/dl.data - data definitions,
dl/lib/dl.vars - environment file,
dl/lib/prog/* - program files in the library,
dl/lib/desc/* - description files for catalog cards,
dl/lib/link/* - link file indexes.

See Also

dl(C)

Author

Jerry Hewett

Name

eduser - Edit a user's account

Syntax

eduser [*name*]

Description

Eduser allows you to change the account information for users on the system, such as names, addresses, and phone numbers. In addition, you may adjust the amount of time allotted for usage each month, the rate per minute at which the user is charged for access, and the number of inactive days allowed before the account expires.

The user editor is very simple to use in that it takes advantage of special terminal functions, such as cursor addressing and other features provided by the terminal installed.

To start *eduser* you can give it a name to bring up for editing (the default is the first user listed on the system alphabetically). *Eduser* will read the password file into memory and then display a form on the screen, filling it in with the specified user information.

At the bottom of the screen, before the cursor, the number of accounts is shown between square brackets (e.g., [128]). Commands at this point are:

Next	Display information for the next user in the database.
Prev	Backup and display information for the previous user.
Find	Search the database for a user name. (The login name, not the user's full name).
Edit	Enter the edit mode to make changes for the current user on the screen.
Dump	Send a copy of the user's name, address, and phone number to the attached printer device in slot 1. (Don't use this if you don't have a printer!)
Quit	Quit *eduser* and return to the shell.

While in Edit mode, any character you type will be inserted at the cursor's position, as long as the field not already filled with the maximum characters allowed. The following editing keys can be used:

RETURN	Move down to the next field.
Control-J	Move down to the next field.
Control-K	Move up to the previous field.

Control-H	Move left one character.
Control-U	Move right one character.
Control-A	Go to the start of the field.
Control-E	Go to the end of the field.
Delete	Erase the character to the left of the cursor.
Control-D	Gobble up the character under the cursor.

Any time you move above the first field, or below the last field, you'll leave Edit mode and return to the main prompt. If any changes were made, they are written to the database at this point for the specified user.

If any changes were made to the ‘‘Full name’’ field for any of the users, the password file will be updated when you quit *eduser*.

If a user is allotted zero minutes of usage per month, the system will regard this as an infinite amount -- meaning, the user has no time limit imposed. Likewise, if the inactive days are set to zero, the user's account will never be removed due to any length of inactivity.

Note

Any accounts which have User-ID's of 0 will bring up record 0 of the account database. This record should not be messed with. Leave it alone.

Only root-level users can run *eduser*.

Diagnostics

‘‘inadequate terminal’’, the terminal does not have functions required for use with *eduser*.

Files

$/etc/adm - account database,
$/etc/passwd - password file.

See Also

adduser(ADM), rmuser(ADM)

Name

finaming - File naming status and control

Syntax

finaming [-ls]

Description

Use *finaming* without arguments to report the operating system’s current naming convention: AppleTalk Filing Protocol or standard ProDOS. AFP naming is desirable over ProDOS’s rigid syntax checking in order work with files on AppleTalk devices.

Using -l (long names) turns on AppleTalk Filing Protocol naming, while -s (short names) turns on standard ProDOS naming.

Note

This command is useful from within the system’s $/etc/rc2 script to enable AFP Long Name convention, assuming AppleTalk services are present.

See Also

rc(ADM)

Name

flip - Flag inactive persons for removal

Syntax

flip [-a] [-d *days*]

Description

Flip (FLag Inactive Persons) scans through the entire user account database ($/etc/adm) looking for *staff* and *guest* accounts that have been inactive for more than their grace period. Any accounts found to be inactive beyond the grace period are written into a file named *rmip* in your $home directory. (If the grace period for an account is zero days, *flip* ignores the account).

The -a option causes *flip* to include accounting information in the report file, listing the user's full name, address, and phone number.

The -d option overrides each user's grace period with a set number of *days* applied to all accounts.

The report file that *flip* generates into *$home/rmip* contains lines showing the flagged user's name, and how many days beyond the inactivity grace has lapsed.

Example:

```
jdoe            # inactive 35 days (max=30)
sjones          # inactive 45 days (max=40)
tch             # never activated
```

The last entry indicates that the user's account was created, but a login attempt was never made. This happens when a new user forgets his password before logging in for the first time.

The *rmip* file is formatted for use with *rmuser*.

Note

If more than ten accounts are flagged for removal, the report file is placed in the operator's directory with the name *rmip.big*. This is a safety feature in the event that the computer's clock leaps ahead so far that many, if not all, accounts are flagged for deletion.

Flip does not remove accounts, it simply flags them for removal.

Files

$/etc/adm - account database,
$home/rmip - report of inactive persons.

See Also

eduser(ADM), maint(ADM), rmuser(ADM)

Name

gid - Group ID utility

Syntax

gid [*option*]

Description

Gid provides the sysop or root level user with fast information for completely tracking all user access on a system.

The command line options select the various types of reports to display:

-r	*root* accounts. Similar to the format of who.
-s	*staff* accounts. Similar to the format of who.
-i	*all accounts* with interpreter used by the login ID. Cannot be used with -p.
-p	show *passwords* in the report. Cannot be used with -i.

Reports can be redirected to file by including >*filename* as the last argument on the command line. The report file contains all access information in one neat little package that can be viewed, edited or printed.

Files

$/etc/passwd - password file,
$/etc/adm - accounting database.

See Also

who(C)

Author

Joel Bressman.

Name

maint - System maintenance

Syntax

maint [-r]

Description

Maint, a shell script that should be invoked as a daily cron task, performs many housekeeping functions. It generates a report of the tasks it completes and mails it to the administrator.

Maint reports on the following:

1. Users who recently called.
2. Newly added accounts.
3. Expired accounts.
4. File server activity.
5. Network mail traffic statistics.
6. Errors during network connections.
7. Errors during news processing.

If the -r flag is included, *maint* removes any accounts that have expired.

When done, *maint* chains to *maint2* if it exists. *Maint2* is the local maintenance script that performs host-specific tasks. These tasks can include the disposition of log files (e.g. rotating, printing, and/or removing them), and running *mdssclean*, *synctime*, or any other periodic maintenance utilities.

Files

$spool/logs/* - system log files.

See Also

cron(ADM), flip(ADM), rmuser(ADM)

Name

mc - Modemcap editor

Syntax

mc [*modemcap*]

Description

Using a full-screen display, *mc* is used to create or edit modem capability (modemcap) files. These files include settings and commands that control external modems for use with ProLine.

When *mc* is invoked without an argument, it reads the system configuration file ($/etc/rsrc/startup.rsrc) and loads the currently active modemcap file. Use cursor keys (or the T-pattern from the 8, 4, 5, and 6 keys on a numeric keypad) to move from field to field on the display. A ‘‘>’’ points to the current field.

A row of ‘‘buttons’’ at the bottom of the display controls file management functions:

[Quit]	Quits *mc*.
[New]	Work with a new modemcap file. This prompts you for a modemcap name. If the name given already exists, it is loaded into the editor. If the file you enter does not exist, a new modemcap is created.
[Save]	Saves changes made to the modemcap. You’re given the opportunity to save it under a different file name.
[More]	(Same as pressing the Tab key). Switches to the alternate modemcap editing screen. The primary screen includes communications settings and timing information. The secondary screen displays modem commands.

To activate a button, press the first letter of its name (e.g., ‘‘Q’’ to quit).

Communication Settings

To change items on the primary screen, use the right and left arrow keys, or 4 and 6 on the numeric keypad. The primary screen items are:

Variable Speed	Determines if the serial port rate should match the modem’s reported connection rate. Set to *Yes* for an old-style modem (typically, 2400 bps or slower). Set to *No* for high-speed modems that operate best at a fixed speed to employ features such as data compression.
High Speed	The highest speed both the modem and computer will accept.
Flow Control	Normally set to *RTS/CTS* if a hardware handshaking cable is used, otherwise, set to *None*.

Has Carrier Detect	Must be set to *Yes* for correct BBS operation.
Error Correction	Set to *Yes* if the modem includes commands to regulate error correction.
ATA Answers	Set to *Yes* unless using an old modem, such as the Apple Modem 300 or USRobotics Password, which cannot answer a ringing line with the ATA command.
Use DTR To Hangup	If the modem is used on an Apple IIGS with a cable that supports hardware handshaking, set this to *No*. Normally, this item should be set to *Yes*.

Timing

Hangup Duration	The time that DTR is held low to force a disconnect (used only if the previous item is set to *Yes*).
Result Code Delay	The time that the system waits for a response from the modem after sending a command.
+++ Guard Time	The delay before and after the modem escape command is sent.
Attention Delay	The delay before a new command is sent after receiving a response from a previous command.

Commands

Items on the secondary screen are:

Main Init	Commands that correctly configure the modem to operate with ProLine. This command normally starts out by setting the factory defaults (&F) and any other commands needed to adjust the defaults follow. ProLine does not require command mode echo, so E0 can be given.
Aux Init	Commands, such as S register definitions to further configure the modem for operation with ProLine. Most factory defaults are fine, except for S7, which should be set to 255 to allow ProLine to control the duration of connection attempts.
Exit Init	The command sent to the modem when the system shuts down.
MNP	The MNP On and Off commands adjust the modem’s error correction feature.
Busy	The Make Busy and Not Busy commands control the modem’s ‘‘hook’’ state. Make Busy takes the phone off hook. Not Busy puts the phone back on hook.

Hangup	Hangup and Post Hangup commands are sent before and after, respectively, a disconnection request.

Files

$/sys/modemcaps/* - modemcap files

See Also

startup(ADM)

Name

mount - Mount disk devices

Syntax

mount *device* ...

Description

Mount restores the selected disk devices into the operating system’s active device list. This has the effect of making unmounted volumes visible again.

A *device* argument consists of a slot and drive specifier (e.g. 5.1 or 5,1 for slot 5, drive 1), or the name of the volume to mount. If only a slot is given, *mount* attempts to mount the devices in both drives for that slot.

Examples:

```
mount /foo /bar
```

Mounts the volumes /foo and /bar.

```
mount 3.2 /ram5 6
```

Mounts the volumes in slot 3, drive 2, /ram5, and both volumes in slot 6.

Information for each unmounted volume is maintained in a resource file. This information is necessary in order to put unmounted volumes back online. The mounting information is valid as long as there have not been any changes in the location of disk devices in the system’s slots. If interface cards move around, the resource file must be deleted.

Diagnostics

‘‘mount.rsrc not found’’ -- the resource file was not found; *unmount* has not yet been run to unmount a volume.

‘‘volume is already online’’ -- the volume specified is mounted.

‘‘volume not in mount.rsrc’’ -- the volume name given is not found in the resource file.

‘‘no mount info for device’’ -- the device given has no mount information; the device hasn’t been unmounted.

Files

$/etc/rsrc/mount.rsrc - mount resource file.

See Also

df(C), unmount(ADM)

Name

notice - Display a notice

Syntax

notice

Description

Notice is used to display a file that resides in the $/etc/notices directory. The file displayed is the name of the command. For example, if the *notice* command were entered at the shell, $/etc/notices/notice would be displayed. If the *notice* program were copied to $/bin/readme, then by typing *readme* in the shell, the $/etc/notices/readme file would be shown.

This program is useful for temporarily replacing other programs so that when a user tries to invoke what appears to be an application, the notice is displayed instead.

Name

rc - Command script for automatic restart

Syntax

$/etc/rc

Description

Rc is a command script that controls the restart process after the system has been rebooted. When the system is restarted and successfully initialized, control is passed to the *rc* script, which runs under *csh*. *Rc* performs its tasks, chains to *rc2* is present in $/etc, and when done, control returns to *login* which waits for system events.

Rc2 typically contains commands that prepare the system for proper operation, such as copying files from fixed disk drives into faster RAM disks, mounting and/or unmounting devices, system maintenance, and so on. *Rc* should not be modified if possible. *Rc2* can be modified as desired.

The runtime performance of many applications can be increased by copying key files from a relatively slow fixed disk into a high-speed RAM disk. The path to a RAM disk is normally assigned to the shell variable ‘‘tmpdir’’. So, *rc2* may begin by copying the following files as shown:

```
echo rc: copying files to $tmpdir
cp $/etc/aliases $tmpdir            # for sendmail
cp $/etc/paths $tmpdir              # for sendmail
cp $/etc/plush.m $tmpdir            # for plush
cp $/bin/plush $tmpdir              # for plush
cp $/etc/cshrc $tmpdir              # for csh
cp $/bin/cshx $tmpdir               # for csh
cp $/bin/csx $tmpdir                # for cs
cp $/sys/modules/parse $tmpdir      # for shells
echo rc: file copying completed
```

Check the manual entries for other applications to see if they take advantage of having their files reside on a RAM disk.

See Also

boot(ADM), csh(C), scripts(M), startup(ADM)

Name

rmconf - Remove a conference

Syntax

rmconf

Description

Rmconf is a shell script that removes a conference and all of its messages from the conference system. It asks for the prefix to the conferencing area (i.e., ‘‘/hd0/cs’’), and then asks for the name of the conference to remove (i.e., ‘‘rumors’’).

Rmconf first removes the descriptive entry of the named conference from the Conference Listing file. You are then asked if you’re ready to remove all the messages in that conference, and if so, type ‘‘y’’ followed by a carriage return. Any other response will cancel *rmconf*.

The entire conference and its associated topics are removed.

Files

$/sys/pcs/cs.list - description file of conferences.

See Also

cs(C), cs.maint(C)

Name

rmtopic - Remove a conference topic

Syntax

rmtopic [*conf/topic*]

Description

Rmtopic is a simple way to remove topics from conferences in the Conference System. It can be invoked as an interactive program, or as a command with arguments

To invoke *rmtopic* as an interactive program simply type *rmtopic* with no arguments. You will be prompted with a list of the current conferences and asked which conference you wish to remove a topic from. If there is only one topic in the conference, *rmtopic* will print an error message and end otherwise you will be shown a list of the current topics and prompted for the name of the topic to remove. *Rmtopic* will then delete the topic. In this mode, *rmtopic* can be aborted by simply pressing RETURN at any prompt or by pressing your cancel key.

If a *conf/topic* argument is supplied, it must be in the correct form -- the name of the conference, a slash (‘/’) and the name of the topic. For example, if you wanted to remove the topic ‘‘stuff’’ from the ‘‘chatter’’ conference the argument would be ‘‘chatter/stuff’’.

In all cases, *rmtopic* will not attempt to remove a topic if it does not exist. *Rmtopic* will not remove the last topic in a conference, instead you will be prompted to use rmconf. Also, *rmtopic* will abort with an error message if the conference specified does not exist.

Note

In some cases, adding and/or removing topics can mess up the user’s last read message pointers.

See Also

cs(C), cs.maint(ADM), csmod(C)

Author

Daniel Davidson. Bugs or suggestions should be sent to danield@pro-grouch.

Name

rmuser - Remove users

Syntax

rmuser [-p] [-f *file*] *user* ...

Description

Rmuser removes accounts from the system's password file, and deletes the user directories and any system files (mail, etc) associated with the *user*. More than one user can be removed by specifying multiple names.

To remove many users, the a list of names to be read from a disk file by using the -f option followed by the name of the *file* containing the list. This file must be formatted such that each line starts with the login name of the user to remove. If there is any more text on the same line following the name, a tab or space character must separate it.

If the -p option is given, progress information is displayed as it works.

Files

$/adm/* - system storage for user's data files,
$/usr/* - user's home directory,
$/sys/mail/* - user's mailbox.

See Also

flip(ADM), maint(ADM)

Name

startup, proline - ProLine configuration and startup

Description

ProLine is launched by running the *Startup* program which resides in the system's root directory ($/). This program automatically senses an active ProLine configuration and engages auto-start mode to bring up the ProLine system. If the system is not configured, it goes directly into the ProLine Installer Menu.

During the auto-start period, the administrator can press ESC to abort auto-start and go to the Installer menu, or press RETURN to immediately launch ProLine. After a short period of no activity, auto-start runs the system in a turn-key fashion by running the *ProLine* program also in the system's root directory ($/).

ProLine begins by obtaining the system configuration from the resource file $/etc/rsrc/startup.rsrc. Then it loads in required system modules, creates any directories not present in the locations in which they're expected, and initializes the modem.

If the modem fails to initialize properly after two attempts, an alarm sounds at the console, and the operator is prompted with:

```
(A)bort, (I)gnore, (R)etry:
```

Choosing (A)bort cancels *ProLine*. (I)gnore tells *startup* to ignore the failed attempt and continue. (R)etry allows *ProLine* to try again. If operator intervention does not occur within 30 seconds, *ProLine* automatically chooses (R)etry.

After initializing the modem, *ProLine* determines if the shell script $/etc/rc exists. If so, *ProLine* launches the C-Shell ($/bin/csh) and has it execute the $/etc/rc script. Typically, this script includes commands that complete the system setup.

Eventually, control is passed to $/bin/login. *Login* handles system events such as answering calls, user logins, cron tasks, and so on.

Startup Resource File

The startup.rsrc file begins with a count-prefixed list of modules required by ProLine. These modules provide ProLine with basic telecommunications abilities and utilities. Example:

```
6
Store|GS
ModemWorks
Time|GS
Serial|GS
Console
Modem
```

The pipe character (|) used before ‘‘GS’’ with some modules allows *ProLine* to automatically select the Apple IIGS version of the module if ProLine is running on an Apple IIGS. See the file $/sys/modules/contents for a description of each module. A count-prefixed list offers the flexibility of loading additional modules.

Following the list comes these items, each on its own line:

modemcap	The name of a modemcap (modem capability) file is given. Modemcap files reside in $/sys/modemcaps. See $/sys/modemcaps/contents for a description of each file.
slot	The slot number of the port to use.
speaker	A code regulating the attributes of the modem’s speaker (3=off, 4=on only during connections, 5=always on).
printer	The name of a Printer module.
printer slot	The slot of the printer.
host name	The name of the host system (e.g. ‘‘pro-sol’’).
domain	The Internet domain by which the host is governed (e.g. ‘‘.cts.com’’). For new systems that have not processed a site map, nor have received a paths database, this line should be left blank.
admin’s login	The login name of the administrator.
admin’s name	The full name of the adminstrator.
time zone	The three-letter time zone.
temp dir	The path to a directory in which temporary files are created and deleted by the system (e.g. ‘‘tmp’’). A RAM disk is perfect for this. Do not use the root directory of any volume, as a volume directory has a limitation of 51 files (the /RAM volume, only 12 files). Always specify a path to a subdirectory on that volume (e.g. ‘‘/ram5/tmp’’).
spool dir	The path to a directory in which logs, mail and network news files are temporarily stored for processing (e.g. ‘‘spool’’). Using a large RAM disk (e.g. ‘‘/ram5/spool’’) can dramatically improve system performance and avoid excess wear on drives. Note, however, that RAM disks that aren’t battery-backed are volatile and can be destroyed in the event of a power failure. A RAM disk may also limit the amount of mail and news that can be processed due to its size, causing system errors. It may be best to choose a directory on fixed media.

ProLine will create the temporary and spool directories if they are not present during startup. Any unqualified paths (ones that do not start with a slash) are assumed to be located in the ProLine root directory.

Files

$/etc/rsrc/startup.rsrc - startup resource file.

See Also

boot(ADM), login(C), rc(ADM)

Name

synctime - Synchronize clock to an atomic time reference

Syntax

synctime

Description

Synctime sets the system clock to the National Institute of Standards and Technology (NIST) Time Mark Generator Device, an atomic standard in Fort Collins, Colorado. *Synctime* synchronizes itself with the service, sets the system clock, and adjusts the time zone (if necessary), all during a 15 second phone call. The service is available on a three-line hunt group at (303) 494-4774. Each of the three lines has its own Time Code Generator Device linked to the NBS 9 cesium beam atomic frequency standard used as a time base at NIST.

This ‘‘atomic clock’’ is accurate to one second in 300,000 years. Since data sent through the phone system takes time arrive, the service employs a communications delay correction feature. This allows the received time to be accurate within milliseconds.

Synctime requires a resource file in the $/etc/rsrc directory called ‘‘synctime.rsrc’’. The resource file’s format follows this example (comments are shown next to each line):

```
1-303/494-4774   # NIST number
1200             # Dialup speed
-8               # Hour difference from UTC
PST              # Standard time zone
PDT              # Daylight savings time zone
1900             # Year 0 of this century
```

Time information is sent in Universal Coordinated Time (UTC) format, also known as Greenwich Mean Time (GMT), the world time at the zero meridian.

The time zone difference you enter will offset the UTC time and date received and set your clock to your local time. In places such as Newfoundland where the time difference contains a fraction of an hour, use ‘‘-3.5’’. If you are in California, use ‘‘-8’’. In New York, use ‘‘-5’’.

The next two lines in the resource file give the three-letter time zone abbreviations for your location, starting with standard time. Some areas of the nation do not make use of daylight savings time, so enter the standard time zone abbreviation on both lines.

The last resource line holds year 0 for the current century (e.g. 1900). This value must be adjusted every 100 years. You may have to adjust it only once in your lifetime.

As *synctime* proceeds, it writes progress information to the console, and keeps a log of the events as they happen. (The log is mailed to ‘‘root’’ when *synctime* is done). *Synctime* reports the number of seconds the clock was off, the newly set time according to the atomic clock reference, and the number of days, if fewer than 50, before a daylight savings or standard time adjustment. It also updates the time zone, if necessary.

Files

$/etc/rsrc/synctime.rsrc - resource file,
$/etc/rsrc/startup.rsrc - holds current time zone abbreviation.

Name

tc - Termcap editor

Syntax

tc [*termcap*]

Description

Tc is used to create or edit terminal capability (termcap) files. It uses a full-screen editor to make editing easy. The editing screen consists of four parts: the termcap code grid, the command menu, the mode line, and the function list.

The termcap code grid holds the raw emulation codes (shown in hexadecimal). Use the cursor keys to move around the grid. The cursor keys are Control-H for left, Control-U for right, Control-K for up, and Control-J for down. You can also use the inverted T-pattern of the 1, 3, 5, and 2 keys on a numeric keypad.

The mode line, below the grid, is the status display. It shows various settings and other information. Sections in the mode line are:

ANSI:	Y = ANSI-style emulation sequences.
	N = non-ANSI emulation sequences.

VT52:	Y = VT52-style screen clear sequence. N = Normal screen clear sequence.
Leadin:	ASCII code (in decimal) of the lead-in prefix.
Row:	Terminal’s home row in decimal.
Curs:	Order of coordinates sent with the GotoXY function, plus the offset added to each coordinate sent.
Pos:	Current position in the grid, shown in hex, decimal, and the corresponding character on ASCII chart.
Sup:	This flag represents character suppression for the ASCII character that corresponds to the current grid position. Y = the character is never displayed on the local console. N = the character can be sent to the local console.
Pfx:	This flag determines if the lead-in prefix is required for the emulation code at the current grid position. Y = lead-in prefix required. N = lead-in prefix not required.

The function list consists of the following:

```
Unused  = no function
GotoXY  = move cursor to X,Y coordinates
ClrScrn = clear the entire screen
ClrEOS  = clear from cursor to end of screen
```

```
ClrEOL  = clear from cursor to end of line
InsLine = insert a line
DelLine = delete a line
InsChar = insert a character
DelChar = delete a character
Home    = move cursor to the home position
Beep    = beeps
CR      = move cursor to start of line
Inverse = turn on inverse mode
Normal  = turn off inverse mode
ScrllUp = scroll screen up
ScrllDn = scroll screen down
Up      = move cursor up
Down    = move cursor down
Right   = move cursor right
Left    = move cursor left
SoftTab = move cursor to next tab stop
HardTab = print spaces to next tab stop
EraseLn = erase entire line
InsOn   = turn on insert mode
InsOff  = turn off insert mode
UndLnOn = turn on underline mode
UnLnOff = turn off underline mode
MsTxtOn = turn on MouseText mode
MsTxOff = turn off MouseText mode
```

Tc recognizes the following single-character commands:

n New -- switch to a new termcap or create one from scratch. You're asked to enter the name of the termcap file. If you enter only a filename (no "/" characters), *tc* assumes that the termcap resides in $/sys/termcaps. Otherwise, *tc* uses the explicit pathname given.

w Write -- save the termcap file. You can use the default pathname, or enter a new one.

q Quit -- exit *tc*.

f Function -- assign a function to the code at the current grid position. The function list entries hilite as the cursor keys are used to move around the list. When the desired function name is selected, press RETURN.

p Prefix -- toggle the lead-in prefix requirement for the code at the current grid position.

s Suppress -- toggle the local console character suppression flag for the ASCII character represented by the current grid position.

' Find Character -- move to the location in the grid that represents the ASCII value of the character entered.

#	Find Decimal -- move to the location in the grid based on the decimal value entered.
$	Find Hex -- move to the location in the grid based on the hex value entered.
a	ANSI -- toggle ANSI emulation sequences. When toggled on, emulation sequences follow the ANSI standard (ESC followed by [and a command letter, plus any arguments). This mode is used for ANSI-BBS and VT100 emulation. Terminals such as VT52 and Televideo 912 do not use ANSI sequences.
c	Cursor Offset -- set the offset added to X and Y coordinates when used in GotoXY sequences. Many terminals add 32 to X and Y coordinates.
h	Home Row -- set the home row. The home row for most terminals is at the top of the display. A few terminals use the bottom line.
l	Lead-in Prefix -- set the ASCII code of the lead-in prefix, usually 27. Some non-ANSI terminals do not use a prefix for commands; they simply use control codes.
v	VT52-Style Screen Clear -- toggles VT52 screen clearing mode. When on, a ClrScrn sequence becomes two separate sequences: GotoXY to the top left corner of the screen, followed by ClrEOS. Most terminals require only a single sequence to clear the display.
x	XY Order -- toggles XY or YX order for GotoXY coordinates.

The arrow keys or numeric keypad keys, as presented earlier, are used for navigation in the termcap code grid and the function list. Pressing Control-L redraws the entire display.

Files

$/sys/termcaps/* - termcap files.

Author

Morgan Davis (mdavis@pro-sol.cts.com)

See Also

setenv(C), stty(C), tset(C)

Name

unmount - Unmount disk devices

Syntax

unmount *device* ...

DESCRIPTION

Unmount removes the selected disk devices from the operating system's active device list. This has the effect of making a volume temporarily disappear.

A *device* argument consists of a slot and drive specifier (e.g. 5.1 or 5,1 for slot 5, drive 1), or the name of the volume to unmount. If only a slot is given, *unmount* attempts to unmount the devices in both drives for that slot.

Examples:

```
unmount /foo /bar
```

Unmounts the volumes /foo and /bar.

```
unmount 3.2 /ram5 6
```

Unmounts the volumes in slot 3, drive 2, /ram5, and both volumes in slot 6.

Unmount maintains information for each unmounted volume in a resource file. This information is used by *mount* in order to put unmounted volumes back online.

Diagnostics

‘‘volume not online’’ -- the volume specified is not presently online.

‘‘cannot unmount active device’’ -- devices in use by the ProLine filesystem may not be unmounted.

‘‘no volume mounted on device’’ -- no volume exists on the specified device.

Files

$/etc/rsrc/mount.rsrc - mount resource file.

See Also

df(C), mount(ADM)

Name

updates - How to install software updates

Description

ProLine updates may include a number of programs (and their manuals). To make transferring many files as efficient as possible, the updated files are stored in an archive created with the ProLine Archive (*par*) program. The update file, ending with a ‘‘.par’’ extension, is distributed over the ProLine network to the system administrator at each site.

To install an update, the administrator follows these three easy steps:

1. Login, and change directories to $/ with this command:

   ```
   cd $/
   ```

2. Invoke the unpar command on the archive (residing in the administrator’s directory):

   ```
   unpar -r ~/update.par
   ```

   The name of the .par file varies from update to update, but always ends with ‘‘.par’’.

3. Reboot the system to begin a new ProLine session:

   ```
   boot -r
   ```

The administrator should move the ProLine update archive offline to a safe place (e.g. a backup disk).

Compressed Update Archives

If an update consists of an enormous amount of data, it is usually compressed using a utility such as ShrinkIt. These updates are sent out with a ‘‘.shk’’ extension and must be decompressed into the resulting ‘‘.par’’ file before the steps above can be taken.

Encoded Update Archives

If an update is distributed with a ‘‘.uu’’ suffix, it indicates that it must be decoded using the *uudecode* program first. A ‘‘.par.uu’’ or ‘‘.shk.uu’’ ending means an archive is contained in the uuencoded file.

See Also

unpar(C), uudecode(C)

ProLine

General Commands

Name

add - Append files

Syntax

add *file... target*

Description

Use *add* to append one or more files to a target file. The source file(s) may be in another directory, in which case a path must be given.

If the *target* is a directory, the source *files* are added to files in the *directory* using their original filenames.

If the *target* file does not exist, it is created, and the contents of the source files are appended.

If a source *file* is given, but no target *file* argument, the current working directory is assumed as the target area, preserving the original filename, though *add* refuses to add a file to itself.

Example

```
add stratos.5 ../mdavis/stratos.novel
```

This adds the file ‘‘stratos.5’’, assuming the current directory is $/usr/brin, to the file ‘‘stratos.novel’’ in the ‘‘$/usr/mdavis’’ directory. If ‘‘stratos.novel’’ was omitted, the original filename ‘‘stratos.5’’ would have been used.

See Also

cat(C), cp(C), mv(C)

Name

alias, unalias - Alias or unalias command names

Syntax

alias [*name value*]
unalias *name*...

Description

Without arguments, *alias* displays a list of the currently defined aliases. With arguments, *alias* associates *name* with the following *value*. The shell applies all transformations on the command line, notably the *value* assigned to the alias. Therefore, care must be taken when using *alias* so that the resulting *value* is stored as desired.

Example:

```
alias   ls      ls -F
alias   bye     "echo It's \$date[4]; logout"
```

Whenever ‘‘ls’’ is entered, the shell expands it to ‘‘ls -F’’. In the second example, ‘‘bye’’ is replaced by two commands. Remember that *values* assigned to aliases pass through all transformations twice: once when the alias is defined, and a second time when the alias is actually used. This includes character escaping, control character insertion, and variable expansion. See csh(C) for a dissertation on avoiding variable expansion when it isn’t desired.

Note that alias only affects the very first argument. If you typed the command ‘‘echo bye’’ you would see ‘‘bye’’ printed on your screen, not ‘‘logout’’.

Argument Insertion

Alias values can contain special characters that indicate where command line arguments should be inserted. They are:

!^	Inserts the first argument
!$	Inserts the last argument
!*	Inserts all arguments

Here’s an example:

```
alias test "echo !\^"
```

Note: The caret character denotes special control code insertion. To include the caret literally, it has to be escaped with a backslash.

Entering ‘‘test’’ followed by one or more arguments causes the shell to first replace it with the ‘‘echo’’ command, and then takes the first argument following ‘‘test’’ and makes it the only argument to the ‘‘echo’’ command. Example:

```
test fizbin data compression
fizbin
```

Alias substitutions may contain multiple argument insertion sequences, and the new command line will be built using the appropriate arguments from the original command line.

Changing and Removing Aliases

Old aliases can be changed with the *alias* command by reentering them with new values.

To remove an alias, use *unalias* followed by one or more alias *names*.

See Also

csh(C)

Name

banner - Print message using big letters

Syntax

banner [*options*] *message* ...

Description

Banner prints a large message to your screen. Each argument is displayed on a new line by itself. To get multiple words and spaces to print on the same line, enclose them in quotation marks.

Options are:

-b	Boldface. The character that makes up the ‘‘pixel’’ of each part of a letter is doubled, giving a wide, boldfaced effect.
-c #	Sets the character that constitutes a letter’s ‘‘pixel’’. By default, the pixel is the pound-sign (#). Using ‘‘-c *’’ would change it to a star.
-m	Monospace. Each letter occupies a fixed width. By default, *banner* displays letters in a proportionally-spaced format.

Example:

banner Stupid!

```
 ###    #                    #       #  #
#   #   #                            #  #
#      ####   #   #  ####   ##     ####  #
 ###    #     #   #  #   #   #   #   #  #
    #   #     #   #  #   #   #   #   #  #
#   #   #  #  #  ##  ####    #   #   #
 ###     ##    ## #  #      ###    ####  #
                     #
```

If the last argument given to *banner* begins with ‘>’, it redirects the output to the filename that follows.

Files

$/sys/modules/banner - banner maker module

Name

basic - BASIC interpreter

Syntax

basic [-l *file*]

Description

Basic invokes the BASIC interpreter. The number of free bytes in memory available is displayed, along with the parent process name of the interpreter which invoked it. This is reported so that the user can reinstate the original shell interpreter when done in BASIC.

If the optional -l flag is used with the name of a BASIC program, the program will be loaded automatically.

Note

Basic can only be executed by root-level users. Once invoked, the ‘‘Unauthorized BASIC Entry’’ feature is disabled so that remote operations in BASIC are allowed. Normally, should the system land into BASIC with the this feature enabled, the system shuts down automatically, terminating the caller’s connection, and reboots.

Warning

It is possible to issue certain commands from BASIC that might crash the system. Internal and external shell commands cannot be executed from BASIC.

Name

calendar - Personal calendar

Syntax

calendar [-e] [*eventfile*]

Description

Calendar reminds you of upcoming events. The events are displayed when *calendar* is called, usually from the login script.

To set up a list of events, you must create a file called ‘‘events’’ in your user directory. Example:

```
DOW  Dy Mon Yr
***  15 Aug **
National Frozen Pizza Day
Wed! ** Sep **
Dart League, 11:30 PM
+10! 31 Oct **
Start working on Halloween costume
***!  7 Dec **
File:daves.birthday
END
```

The first line *must* be included exactly as shown, or *calendar* will ignore the entire file. Each event entry consists of two lines, a date line and a message line. The date line can include wildcard characters for a ‘‘don’t care’’ situation. The date line must be formatted exactly as shown, or that event will be ignored. A maximum of 99 events can be stored.

In the example above, the first event is to occur on August 15th, with the day of week and year ignored. If *calendar* is called on August 15th, the message ‘‘National Frozen Pizza Day’’ will be displayed.

If *calendar* is called on any Wednesday in September, the second event is triggered, and the message ‘‘Dart League, 11:00 PM’’ is displayed.

The third event is an example of look-ahead. The first character on the date line is a ‘‘+’’, followed by the number of days advance notice required (up to 99). In the example, the message ‘‘Start working on Halloween costume’’ will be displayed every day between October 21 and October 31, along with a countdown of the number of days until the event occurs. After October 31, the message will no longer be displayed. Wildcards may be used in combination with look-ahead. When wildcards are used with look-ahead, *calendar* will check the following month or year if appropriate. For example:

```
+10!  5 *** **
Pay phone bill
```

This event would be displayed from about the 25th of the month (depending on which month) through the 5th of the following month. If the current date is December 31, ‘‘5 days until:’’ will be displayed, since *calendar* will look past the end of the year.

The last three examples show a feature new to v1.5 - auto-delete. If the fourth character of the date line is a ‘‘!’’, the event will be saved after it is displayed. If the ‘‘!’’ is missing, the event will be deleted after it is displayed. For look-ahead events, the event will not be deleted until the actual date occurs.

The last line of the file must be ‘‘END’’ by itself, or calendar will give an error.

The ‘‘events’’ file can also be created and edited with the built-in editor. To use the editor, invoke *calendar* with the -e parameter. The editor is quite self-explanatory.

Version 1.6 includes an improved editor and adds two new features: alternate eventfile, and textfile display.

The alternate eventfile works just like the main one, except the filename must be used when calling *calendar*. Simply use ‘‘calendar alt.events’’ or ‘‘calendar -e alt.events’’. The alternate events file may use any legal filename, and there is no limit to the number of alternate files.

The textfile display feature allows you to display any textfile as the message for a given event. This makes it easy to add multiple events for a particular day, or to make a special sign. To use this feature, enter ‘‘File:filename’’ as the message text for that day (as shown in the fourth example).

Files

$/usr/*/events - User’s default event list.

Author

David Zachmeyer. Bug reports or suggestions should be sent to djz@pro-phc.

Name

calls - Report system usage

Syntax

calls [*options*]

Description

Calls displays useful statistics about the system. Depending on the options selected, a variety of reports can be generated.

Options are:

-f file	Process the syslog file specified by the *file* argument instead of $spool/logs/syslog.
-u	User report. Provides information for all users that have called the system.
-s	Statistics report. Shows statistical information such as idle time, percentage of calls based on baud rate, percentage of system uptime used by certain baud rates, etc.
-t	Time report. Draws a bar chart for the current 24 hour period, showing the frequency of usage during each hour of the day.

If no report type is specified, *calls* displays all reports.

Notes

Output can be redirected to a file if >*file* is given as the final argument.

Calls cannot process syslog files that go into another year.

Files

$spool/logs/syslog - file containing daily login entries

See Also

log(C), uutraf(NET)

Name
cat - Concatenate and print

Syntax
cat *file* ...

Description
Typically, *cat* is used to show the contents of a *file*. Any type of file can be displayed, but files containing ASCII text produce the only meaningful results.

Example:

```
cat login
```

This displays the contents of your *login* script. More than one file can be shown by giving *cat* a list of file names.

Cat can be used to redirect a file to another file or device, including appending output to existing files (hence, *concatenation*). Examples:

```
cat net.report >.printer
cat intro userlist contacts >>memo
```

The first example sends the net.report file to the printer device. The second example appends the files intro, userlist, and contacts, to the end of memo.

See Also
grep(C), lpr(C), more(C), tail(C)

Name

cd - Change working directory

Syntax

cd *directory*

Description

Cd is used to change directories. After logging in, the working directory is set to your home directory. From there, you can change to any directory by typing:

```
     cd directory
```

If *directory* is a partial path (does not begin with the root directory), *cd* looks downwards from where you are for the directory to change to.

Short Cuts

cd	Without arguments, *cd* puts you in your home directory (same as typing ‘‘cd $home’’).
cd /	Changes to the root directory.
cd $/	Changes to the ProLine system directory.
cd ..	Changes to the parent directory of the current working directory.
cd ~/bin	Changes to the bin directory residing in your home directory. (The tilde character is the same as $home).

See Also

pwd(C)

Name

chat - Chat mode

Syntax

chat [-r *rings*] [-t] [-v *volume*]

Description

Chat lets a caller and the system operator carry on conversation in real time. Upon invoking *chat*, an audible signal (much like a telephone) rings on the host computer. If the operator is available, the chat request is serviced, placing the user and the operator into a chat mode. If the operator is not available, the user is given the chance to send an electronic mail letter instead.

Depending on the type of terminal the user emulates, one of two chat modes may be invoked. Both modes offer word wrap and limited text editing.

With terminal emulation, a split-screen mode is activated. The user and operator have their own windows in which to type messages unencumbered by the other's input. In fact, each may type at the same time without interrupting the other. Typing Control-L redraws the entire screen. Typing Control-X clears the user's or operator's assigned window. Pressing the Escape key exits the *chat* program.

If no terminal emulation is set up, a regular chat loop is activated. In this mode, the user and operator must take turns entering lines of text so as not to ‘‘step on’’ one another. Standard chat protocol in this method involves pressing RETURN enough times to put at least one blank line after a typed message is completed. This signals the other user that a response is expected. Entering a period alone on a new line exits the *chat* program.

The options are:

-r N	Rings (default = 6), where N is the number of ‘‘phone rings’’ that sound before *chat* gives up.
-t	TTY mode (no split-screen emulation), even if adequate emulation is enabled.
-v N	Volume level (default = 7), where N is a number from 0 (quiet) to 15 (loudest) which controls the loudness of the ringing signal at the console. Only the super user may specify this option.

Operator's Note

To regulate chatting times, hooking up a joystick or game paddle to the computer provides an ‘‘availability switch’’. Moved to the left, chatting is disabled (you have ‘‘left’’). Moved to the right, chatting is enabled (you are ‘‘right’’ there).

Name

cp - Copy a file

Syntax

cp *file*... [*target*]

Description

Cp copies one or more files to another file or directory. The source file may be in another directory, in which case a path must be given.

If the *target* is a directory, the source *file(s)* are copied to files in the *directory* using their original filenames.

If the destination file does not exist, it is created, and the contents of the source files are copied.

If a source *file* is given, but no target *file*, the current working directory is assumed as the target area, preserving the original filename, though *cp* refuses to copy a file to itself.

Example:

```
cp $/usr/jholt/rocks garbage
```

This copies ‘‘rocks’’ from $/usr/jholt to the file ‘‘garbage’’ in the current working directory. If ‘‘garbage’’ was omitted, the original filename ‘‘rocks’’ would have been used.

See Also

add(C), cat(C), mv(C), rcp(NET)

Name

cs - Conference System

Syntax

cs

Description

The *cs* command invokes the conference system. It is called a ‘‘conference system’’ instead of a ‘‘bulletin board’’ or ‘‘message base’’ because active participation by system members is encouraged.

The conference system is where system members gather to discuss a variety of topics in open or closed forums. Networked conferences can include offsite participants, expanding the forums to users on computer systems around the world.

This tutorial will help you master the operation of CS in just a few minutes. First, all you really need to know to get around in the conference system is the RETURN (or ENTER) key. When you press RETURN, you are taken from one unread message to the next, from one topic to another, and so on.

Of course, if you find yourself wondering what to do next, you can type the question mark key (?) and a list of available commands or responses will be shown.

About Conferences and Topics

Each conference is dedicated to the discussion of one particular subject. For example, music. Within the ‘‘music’’ conference, there may be one or more subconferencing areas, called *topics*. Topics for the ‘‘music’’ conference might be ‘‘concerts’’, ‘‘discs’’, ‘‘reviews’’, and so on.

Conferences come in two flavors: *Open* and *Closed*. An Open conference is one that anyone can join, while a Closed conference is by invitation only.

There are two kinds of topics, too: *Local* and *Networked*. Local topics are those in which only local system members participate. Networked topics let the whole world in on the act! People from remote sites, sometimes as far away as the other side of the world, exchange messages in networked topics.

Getting Started

Your first step in using the conference system will be to join conferences that interest you, then read some or all of the existing messages, respond to them, and add your own to the discussion.

There are three main areas in the conference system:

cs>	Select and work with conferences and topics.

read>	Read messages and switch topics in a conference.
add>	Enter text to create a new message.

You can get a detailed list of commands at each prompt by pressing ‘‘?’’.

Joining A Conference

When you first enter CS, press RETURN to join the first conference with new messages since your last visit. If there are no new messages, you can choose to join a conference by using the (J)oin command.

Reading Messages

Once you’ve joined a conference and entered into a topic area, press RETURN to being reading new messages. Popular commands at the read> prompt are:

Next Display the next unread message.

Next takes you from one unread message to the Next. When you run out of unread messages in one conference/topic, Next takes you to the next conference you’re joined to with unread messages.

Skip Change your unread message pointer.

If you’ve joined a new conference that has an over- whelming number of unread messages, you may choose not to read them all. Use Skip to skip over messages you don’t want to read.

Add Add a new message to the conference/topic.

Add allows you to add a brand new message. You’re asked to describe the subject of your message, and then you can begin writing it. This takes you to the add> prompt.

Reply Reply to a message.

Like Add, Reply lets you contribute to the discussion. Use this command when you want to continue an ongoing discussion.

There are many more commands than those listed above. Type ‘‘?’’ at the read> prompt to see the complete listing of available commands.

Adding or Replying

When you Add or Reply to a message, you’re asked to enter the ‘‘Subject’’ of your contribution. Type in a brief description that summarizes what your message is about.

Next, type in your message. As with a pad of paper, you can write out your comment and edit it before you add it to the discussion. If you make a mistake in the line you are presently typing, you can backspace and correct it. If you want to make a change in an earlier line, finish the note and edit it later.

When done entering text, type a period on a new line and press RETURN. This takes you to the add> prompt where you can save or edit your message, as well as other options.

Now You're Ready!

You now know enough about CS to use it effectively. Remember that you can always use ‘‘?’’ if you need to see a list of available commands no matter where you are. If you ever get into a situation that you wish you could get out of, use your Cancel Key (usually Control-C). Call upon the system administrator or one who is knowledgeable with the operation of the system if you find you need further assistance.

Quick Start Checklist

Upon entering CS, you may want to:

1. Use (L)ist to see a listing of available conferences.

2. Then use (J)oin to subscribe to a conference that interests you.

3. Finally, just keep pressing RETURN to read each unread message in succession until there are no more.

Happy Conferencing!

Files

$/bin/csx - CS core program (loads faster from a RAM disk),
$/sys/pcs/cs.herald - Initial greeting upon entering *cs*,
$/sys/pcs/cs.list - list of conferences with descriptions,
$/etc/rsrc/cs.rsrc - CS resources,
$/etc/default/csdata - list of default conferences users join,
$/adm/*/csdata - data files for each cs member,
$/etc/help/* - help files for CS commands.

See Also

cs(F), cs.maint(C), csmod(C), postnews(C)

Name

csh - Command shell

Syntax

csh [-c] [*command*]

Description

Csh is a command interpreter. Without any arguments, *csh* goes through a normal startup procedure, eventually giving you a command line where you can enter more commands.

If *csh* is invoked with a *command* argument, it processes that command and exits when finished. If the -c flag is included, *csh* processes the command, but does not exit. If only -c is given, the shell immediately presents a command line for entering commands (it does not run your *login* script, described later).

About The Shell

Csh is the central program from which all other programs can be accessed. It is often called a command interpreter, acting as a liaison between you and the system. To instruct the system to perform a task, you enter your request on a command line. The shell processes the command line to carry out your request by means of a built-in shell function, or by selecting a program external to the shell to complete the request. The shell contains many internal functions for use in the shell environment. There are even more external programs that can be called upon from the shell. After a command line is processed, the shell prompts you for additional commands until you're ready to quit.

The shell can also process a series (or *batch*) of commands contained in a file, as if they had been entered by hand on the command line. Files containing shell commands are called *shell scripts*. Scripts use special features to control the execution of commands in a way that reduces arduous tasks into simple, automated procedures. More on scripts later.

The Command Line

Executing a built-in or external command from the shell is simple -- just type in the command's name. While typing in a command line, various editing keystrokes are available:

^X	cancel what you've entered so far
^H/DEL	rubout the character left of the cursor
^W	rubout the word left of the cursor

Command Line History

The shell maintains a history of the 20 most recently entered commands. You can scroll through command lines entered and repeat one by pressing Control-P (for previous) to move backward in time toward the oldest command lines. Pressing Control-N (for next) moves forward to the most recent command lines.

The shell also recognizes the exclamation point (!) character at the beginning of each line as a way of referencing specific history lines.

!!	Selects the last entered command line.
!#	Selects a command line by the number associated with #.
!text	Selects the most recent command line that begins with a pattern of text.

Arguments and Special Characters

In many cases, you may need to give extra information, or *arguments*, to a command. To help the shell determine which items are commands and arguments, separate them with space characters.

To specify a single argument that may contain spaces within it, enclose the argument in quotation marks. Example:

```
echo "This is one argument"
```

The shell ignores most control characters entered on the command line. To insert a control character, use a caret before the corresponding letter (e.g., ‘‘^G’’ for Control-G).

If you need to enter a quotation mark or caret in an argument, prefix it with a backslash ‘‘\’’ character. Likewise, to enter a backslash, prefix *it* with a backslash (in other words, two backslash characters make one). Examples:

```
echo "\"Wow!\" she exclaimed."
echo "Here is a caret: \^"
echo "Here is a backslash: \\"
```

Using a backslash to insert special characters is known as *escaping*. There are other characters the shell recognizes as having special meanings, such as the dollar sign ($), tilde (~), period (.), and asterisk (*). These are discussed later.

Built-In Commands

The shell has many built-in commands. They are listed below along with the arguments they require. Optional arguments are shown between [brackets]. Arguments shown with ‘‘...’’ following means ‘‘one or more of the same’’.

add src... [targ]	Adds source file(s) onto the end of the target file. If one file argument is given, the file must reside outside of the current directory and will be appended to a file in the current directory

	with the same name. If more than one argument is given, the final argument is the target file, and all others are appended onto it.
alias [old new...]	Substitutes a command name with a new name (and any additional arguments). The new replacement arguments may contain the following special characters: !^ (inserts the first argument from the command line here), !$ (inserts the last argument here), and !* (inserts all arguments here). Entering alias without any arguments displays a list of the current aliases.
cat file...	Short for ‘‘concatenate’’. *Cat* is normally used to display the contents of a file, but is also useful for redirecting a file to other files or devices (e.g. a printer).
cd [dir]	Changes the current working directory. If the argument is omitted, the current directory becomes your $home directory.
clear	Clears the screen.
cp src... [targ]	Copies a file. If only one file is given, it must reside outside of the current directory, and is copied to a file of the same name within the current directory. If more than one argument is given, the source files are copied onto the target file. If the target is a directory, the files are copied into the directory, retaining their names.
echo [-n] args...	Echoes any arguments that follow. If the -n option is given, no newline is printed after the final argument. To redirect output from echo to a file, include >file as the last argument. To append output to an existing file, use >>file.
exec command...	This replaces the current shell with a new command interpreter. If called from a shell script, the script is terminated.
exit [n]	Tells the shell to quit and return to the calling process. Typing Control-D is recognized as the *exit* command, too. The optional number argument useful for returning a result code to the calling process (default is zero).
history	Displays the 20 most recent command lines.
if expr then [...]	Conditional test to execute commands. This is normally used within shell scripts and is discussed later.
logout	Terminate connection immediately.
mkdir dir...	Makes a subdirectory.
mv src... [targ]	Moves a file from one place to another. It works identically to *cp*, except it deletes the source file after copying it to the target. It is also used to rename a file if both src and targ reside in the same directory.

pwd	Prints out the name of the current Working Directory.
read var	Reads input from the user and assigns it to a variable.
rm file...	Removes files and empty directories.
set [var=value...]	Sets shell variables to specified values. Set without arguments displays the currently set variables.
shift [var]	Shifts the arguments in the command line down by removing the first argument. Without arguments, *shift* affects argv (see the discussion on variables later for details). By including an optional variable name, that variable is affected. This command is typically used by shell scripts for argument parsing.
source file	Executes shell commands from a text file.
unalias name...	Removes aliases.
unset var...	Removes variables.

Many more commands can be called up from disk by the shell. One such command is *ls*, which lists the contents of a directory. To see the names of all the disk-based commands, enter:

```
ls $path
```

Entering a command without any arguments usually displays a line of ‘‘usage’’ information. You can get more information about a command by using the *man* command (short for ‘‘manual’’). *Man* requires the name of the item you want more help on as an argument. It prints out nicely formatted pages from the online manual. (Try the ‘‘man man’’ command -- it lets *man* tell you more about itself).

Access Permissions

The shell watches certain requests to make sure the user has the required access permissions. There are four types of file access: reading from a file, writing to or creating a file, destroying a file, and executing a file. Under ProLine, access permissions are specified for the entire contents of a single directory level, rather than on a file-by-file basis. (See *dstat* for more details).

Output Redirection

If you’re using the system from the local console, you can redirect the output from the shell and other programs to a printer attached to the computer. To do this, include *>.printer* as the final argument to a command line. This tells the shell to turn on printer output. It’s only good for one command, so there is no additional command to turn it off; it will do so when it wants another command from you.

Example:

```
          echo This is a test >.printer
```

Another form of output redirection is to send output into a file, rather than to the console or printer. The syntax is similar to *>.printer*.

Example:

```
          man csh >temp
```

In this case, it is *man*'s job to handle redirecting its output into the file called ‘‘temp’’ (or whatever name you provide). Unfortunately, not all programs on the system provide this ability. Check with *man* to see if a certain command provides file output redirection.

Directory Shorthand

The shell looks at the arguments you give to see if any start with special characters that reference directories. These characters are one or two periods (.), and the tilde (~).

When the shell encounters a single period, it replaces it with the path to the current working directory. If it sees two periods, it replaces them with the name of the parent directory of the current working directory. In other words, two periods means to step back one directory level. So, if a single periods refers to $/usr/jholt, then two periods refer to $/usr.

The tilde (~) is directory shorthand for referencing home directories. To access a file in your home directory called stuff when the current directory is not your home, use ‘‘~/file’’. The shell sees ‘‘~/’’ and knows that you’re referencing your home directory (which is shorthand for ‘‘$/usr/your*login*name/file’’). By introducing the login name of another user between ~ and /, you reference that user’s home directory. For example, ‘‘~jsbach/sonata’’ is short for $/usr/jsbach/sonata.

Shell Variables

Shell variables are short names given to command line items that are cumbersome or hard to remember. The shell comes with a number of preset variables containing useful information. To obtain the value of a variable, a dollar sign ($) is placed before its name (e.g. *$path*)

Variables the shell defines for you are:

/	Directory in which ProLine resides on the local file system. Since one ProLine system may have a startup prefix that is different from another ProLine, the *$/* variable allows ProLine systems to work consistently in the shell from one system to the next.

argc	Contains the count (+1) of the arguments passed to a shell script.
argv	Contains the shell script's name and arguments passed.
caller	The system caller ID number.
cwd	The current working directory.
date	The current date and time in the standard ProLine date format: Sat, 18 Apr 92 10:22:19 PDT
editor	The name of the editor used to edit text files on ProLine.
exit	If set, any errors in a shell script cause the script to halt. By unsetting this variable, errors will allow a script to continue processing.
gid	Your group-ID number.
home	Your home directory.
host	The name of the host system.
login	Your login name.
mail	The path to your mailbox file.
name	Your first and last name.
path	A list of paths to directories containing external command files.
prompt	Your shell prompt.
shell	Path to the shell program.
shlvl	The number of nested shell levels.
status	The result of the last command.
time	The current day of the week, date, month, year, hours, minutes, seconds, and day of week number in this format: Sat 18 04 92 10 22 19 6 Month values are 01 for Jan up to 12 for Dec. Day of week values are 0 for Sun up to 6 for Sat. All numeric values (except for day of the week number) are padded with zeroes to fill out to two digits.
tmpdir	Path to a directory for temporary file storage (usually a RAM disk or fast hard disk volume).

tty	Contains ‘‘modem’’ or ‘‘console’’ depending on how you logged in -- remotely or locally.
uid	Your user-ID number.
version	The shell’s version information.

Subscripted Variables

You can access individual words within a variable by including a subscript after the variable’s name. For example, to access the first word of the $time variable (the day of the week), use $time[0]. To access the hours, use $time[4]. To access arguments passed to a script from within the script, use $argv[x] as appropriate.

Avoiding Variable Expansion

The shell always expands variables to the values they contain, even when placed inside quotation marks. To cause the shell to pass a dollar sign and a word through unexpanded, prefix it with a backslash. (e.g. ‘‘\$time’’). Consider the following:

```
alias   bye   "echo Login: $date[4]^MLogout: \$date[4]; logout"
```

In this alias definition, ‘‘bye’’ is defined with an echo command that includes the $date variable in two places. The first instance is not escaped by a backslash, and therefore when this *alias* command is executed, the current time is inserted. The second instance of $date *is* escaped by a backslash, and therefore the word ‘‘$date’’ is literally inserted into the definition. When the bye alias is actually used, only then would the second instance of $date be recognized (since the backslash would not be present), and the current time inserted. As a result, the bye alias displays both the login and logout times before disconnecting.

The same is true when assigning variables to a variable. If you want your shell prompt to display the current working directory, use:

```
set prompt="\$cwd> "
```

If the $cwd variable reference had not been escaped with a backslash, this command would assign the prompt variable a value containing the current working directory at the time when set was issued. The prompt would never change even as new directories were chosen.

Shell Scripts

You can instruct the shell to execute more than just one command at a time by entering command sequences separated by semicolons (;) on the same line. But, if you need to execute many, often repeated command sequences, it is best to put them into a shell script.

A shell script contains any commands you would enter on the command line. Once created using any text editor, you can execute the script by using the *source* command. (To make a script executable without having to use *source*, change its file type to CMD with the *setfile* command).

Initialization Scripts

When first launched, *csh* sets up an internal environment which can be further customized through various initialization scripts. If it exists, *csh* invokes a system-wide initialization script *$/etc/cshrc*. This script is set up by the administrator to perform various sign-on tasks, like reporting new system bulletins (*news*), reporting mail, and finally invoking the user's personal *login* script (discussed next).

When interactive subshells are launched without arguments, *csh* does not execute *$/etc/cshrc* nor the user's *login* script, but will attempt to execute the user's personal *cshrc* file, if it exists in the home directory.

The $/etc/cshrc Script

The system-wide startup script, *cshrc*, is expected to reside in $/etc or the configured temporary files directory ($tmpdir), which might be a high-speed RAM disk. If the system-wide *cshrc* does not exist, *csh* attempts to execute the user's *login* script during a login session.

The Login Script

Each user has a special script of startup commands. The script is called *login* (~/login). This file can contain a variety of shell commands, and can be modified as desired, allowing a unique and personal entry into the system for each session. Once the *login* script has done its job, the shell may prompt for further commands, but typically *login* invokes the user's personal *cshrc* file.

The *login* script is derived originally from the default script ($/etc/default/login) when an account is first created.

Personal cshrc Script

The personal *cshrc* file (~/cshrc), contains commands that are used to install aliases and set variables such that each shell is set up as the user desires. This script is normally launched by a command in the user's ~/login script, but is automatically run whenever *csh* invokes an interactive subshell.

Script Control Using the IF Directive

To control execution of script commands, use the *if* directive. *If*, and its partners *else* and *endif*, are used to test certain conditions and execute only the commands which should be processed based on the results of those conditions. The condition is determined by the logical expression given to *if*.

An example:

```
if $a = $b then echo TRUE else echo FALSE
```

Another example:

```
if $time[0] = "Sun" and -f workfile then
    echo It is Sunday AND the workfile exists!
else
    echo EITHER it is not Sunday
    echo OR the workfile does not exist.
endif
```

For more details on *if* statements, see the if(C) manual entry.

Files

$/bin/csh - C-Shell initialization program,
$/etc/cshrc - system-wide initialization script,
$/bin/cshx - C-Shell kernel code,
$/usr/* - your ‘‘home’’ directory (~),
~/login - your login startup script,
~/cshrc - your shell initialization script,
$/sys/modules/parse - parsing tools,
$tmpdir/cshenv* - temporary environment file,
$tmpdir/* - C-Shell runs faster when $/bin/cshx, $/etc/cshrc, and $/sys/modules/parse live on a RAM disk.

Author

Morgan Davis (mdavis@mdg.cts.com)

See Also

if(C), plush(C), rc(ADM), scripts(M)

Name

csmod - Conference System moderator’s utility

Syntax

csmod *conference*

Description

Csmod is a tool for conference moderators. It requires the name of a conference as a startup argument. Only the Super User or the assigned moderator of the conference can use *csmod*.

Csmod allows changes to the conference as well as to its topics. The primary command level is for conference-related changes and reports. Commands here are:

<T>	Topic changes. Presents a list of topics in the conference from which one must be chosen to perform changes on a topic. Commands at this level are described later.
<M>	Moderator change. Allows reassignment of a moderator.
<I>	Info file changes. Invokes the editor on the conference’s ‘‘info’’ file in order to create or change it.
<O>	Open status change. Changes a conference’s Open status. Open conferences can be joined by all system members. Closed conferences can only gain users when the moderator manually joins them.
<J>	Join a user. Selectively joins a user to the conference. This is mostly used with closed conferences.
<U>	Unjoin a user. Removes a user from the conference.
<S>	Subscriber report. Displays a list of all system users joined to the conference.
<Q>	Quit *csmod*.

When making changes to a topic with the <T>opic option, a second command level is invoked. Commands here are:

<T>	Topic name change. Renames a topic.
<N>	Network changes. Assigns or changes the network e-mail address to which conference postings are sent. A topic with a blank e-mail address is a non-networked topic. This command also allows you to determine if ‘‘signature’’ files should be attached to outgoing messages.
<I>	Info file changes. Invokes the editor for the topic’s ‘‘info’’ file in order to create or change it.

<R>	Read-Only change. Adjusts the read-only or read-write status for the topic.
<Q>	Quit topic changes. Returns to the main conference command level.

See Also

cs(C), cs(F), cs.maint(ADM)

Name

ctime - Connect time and accounting info

Syntax

ctime

Description

Ctime displays information about your connect time. The current time, the time you logged in, and the previous login time are shown, along with the total number of hours used for your current session, the current month, and previous month, and the total hours spent online since your account was opened. It also tells you how many times you have logged in, as well as the total number of days that you can go without logging in before your account is automatically removed due to inactivity. Current and previous month's access charges are also shown, if any.

Operator's Note

If *ctime* is invoked from the '!' command in *login*, the account statistics for the system will be shown, displaying amassed logins and hours since the system went online.

Files

$tmpdir/utmp - current session accounting,
$/etc/adm - account database

See Also

log(C)

Name

df - Disk free space

Syntax

df

Description

Df prints out the amount of free space on all of the mounted file systems. The reported numbers are in 512-byte disk blocks. The percentage of disk space used is also given.

A sample display looks like this:

```
Dev  Blocks   Free   Used  Used%  Volume
-----------------------------------------------
3.2     127    100     27    21%  /ram
7.1   41616  33302   8314    20%  /a
6.1     280    185     95    34%  /z
```

Df supports output redirection into files.

See Also

du(C), mount(ADM), unmount(ADM)

Name

dl - Data Librarian

Syntax

dl

Description

(Throughout this manual entry, *dl* denotes the location of the Data Library).

The data librarian offers an easy way to upload and download programs. The library metaphor is used extensively throughout this program. Users of the data library can ‘‘open’’ specific card catalogs (a program category; ie., Business, Games, Graphics, Utilities, etc.) and ‘‘browse’’ through the descriptions for each title on the ‘‘shelf’’, selecting which files they want to ‘‘reserve’’ (flag) for ‘‘check out’’ (download) when they’re ready to leave.

Each file in the library has an ‘‘index card’’ describing the file name, type, size, author, and a short synopsis of what the file is used for. The user can search through all index cards in a particular card catalog using ‘‘key words’’ in the title and/or description (ie., ‘‘word’’ would display all index cards that have the search pattern anywhere on the card -- wordprocessing, PC-Word, wordwrap, and ‘‘the latest word’’ would trigger a match). At the bottom of each card is a ‘‘cross-reference index’’ that links the card to various card catalogs in the library, so that only one card (file) has to be ‘‘donated’’ (uploaded). This method makes it easier for the user to find a particular file, and makes multiple uploads into different file areas unnecessary.

XMODEM protocol is used for transferring data files, although if the file is text (TXT) the user can request an ASCII text transfer and capture the file to disk or a copy buffer.

The data library program is highly interactive, and contains an excellent built-in tutorial (the ‘‘information desk’’), so the best way to learn it is to use it.

Files

dl/lib/prog/*	- program files in the library,
dl/lib/desc/*	- description files for catalog cards,
dl/lib/link/*	- link file indexes,
dl/lib/dl.vars	- environment file,
dl/lib/dl.data	- data definitions,
dl/lib/dl.user	- last online database for See What’s New,
$/sys/pdl/dl.dir	- working prefix for the data library,
$/sys/pdl/dl.sort	- DL code overlay for data/pointer sorting,
$/sys/pdl/dl.xfer	- DL code overlay for upload/download,
$/sys/pdl/*	- text files for data library online help.

See Also

dl.maint(C), rx(C), rz(C), sx(C), sz(C)

Author

Jerry Hewett

Name

dstat - Directory access status

Syntax

dstat [*opcode-permission*] [*directory*] ...

Description

Without any arguments, *dstat* displays the directory access status for the current working directory. By giving one or more directory names as arguments, *dstat* reports their associated access statuses.

(The access status pertains only to other users who do not own the specified directories. This is because the owner of a directory holds full access for its contents.)

When a directory is first created, it has no access permission levels assigned. By using *dstat* with *permission* flags, however, the following statuses can be set or cleared:

r	Read permission - viewing, searching, downloading, and copying to other files.
w	Write permission - creating files (but not modifying existing files).
d	Delete permission - complete removal and destruction of files.
x	Execute permission - the ability to execute programs and shell scripts.

To make changes, prefix the permission characters with an opcode as follows:

-	Removes permissions
=	Sets absolute permissions
+	Adds permissions

Examples:

```
dstat +rw      # add read and write access
dstat =x       # set only execution permission
dstat -d       # remove delete permission
dstat +        # (same as = alone) grant full access
dstat -        # remove all permissions
```

You can give more than one permission at a time by including multiple permission characters after the opcode. If permission characters are omitted, full access is either granted (+/=) or removed (-).

When *dstat* is told to adjust a permission level, it assumes the current working directory if no *directory* argument is given. Multiple directories can be set to the same permission levels by specifying more than one directory in the command line.

IMPORTANT

The system administrator should *NOT* allow both *WRITE AND EXECUTE* permission in a directory outside of the $/usr hierarchy! Doing this would allow a user to upload a program or binary file, execute it, and cause potential havoc, catastrophic event, cataclysmic disaster, or worse. The shell limits execution to shell script files by non-root users within the $/usr hierarchy. Therefore, write+execute permission is safe here.

Programmer’s Note

Directory permission settings are stored in the lower 4 bits of the auxiliary field in the directory’s file information. Arbitrarily changing these bits could have disastrous effects should it allow for ‘‘destroy’’ permission in a system directory. The bit assignments are:

```
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|  |  |  |  |  |  |  |  |  |  |  |  |  | x| d| w| r|
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
 15 14 13 12 11 10  9  8  7  6  5  4  3  2  1  0
```

Auxiliary Field in Directory Files

Bits 4 through 15 are reserved for future use.

See Also

mkdir(C), setfile(C)

Name
du - Summarize disk usage

Syntax
du [-a] [-s] [*name*...]

Description
Du gives the number of disk blocks contained in all files and (recursively) directories within each directory specified by the *name* arguments. If no *names* are given, the current directory is assumed.

The optional argument -s causes only the grand total (for each of the specified names) to be given. The optional argument -a causes an entry for each file. Absence of either causes an entry to be generated for each directory only.

Notes
Non-directories given as arguments are not allowed.

This utility reports sizes in 512-byte blocks.

Output redirection is supported only when the -a argument is *not* given.

See Also
df(C), ls(C)

Name

echo - Echo text arguments

Syntax

echo [-n] [*arg*...]

Description

Displays text arguments to your terminal. Spaces are automatically placed between arguments. Normally, *echo* will send a newline after printing the last argument, but it can be suppressed by including -n as the first argument.

Examples:

```
        echo Hello, World!
```

Displays "Hello, World!" followed by a newline.

```
        echo -n Wait...
```

Displays "Wait..." without a newline.

Output from *echo* can be directed to a file by including ‘‘>file’’ as the final argument. Appending is also supported by using‘‘>>file’’.

Echo is an excellent diagnostics command for tracking the execution of shell scripts.

See Also

csh(C), scripts(M)

Name

ed - Text editor

Syntax

ed *file*

Description

ed is the standard UNIX text editor. When the file argument is given, *ed* simulates an e command (see below) on the named file; that is to say, the file is read into ed's buffer so that it can be edited. *ed* operates on a copy of the file it is editing; changes made to the copy have no effect on the file until a w (write) command is given. The copy of the text being edited resides in a temporary buffer. There is only one buffer.

Commands to *ed* have a simple and regular structure: zero, one, or two addresses followed by a single-character command, possibly followed by parameters to that command. These addresses specify one or more lines in the buffer. Every command that requires addresses has default addresses, so that the addresses can very often be omitted.

In general, only one command may appear on a line. Certain commands allow the input of text. This text is placed in the appropriate place in the buffer. While *ed* is accepting text, it is said to be in input mode. In this mode, no commands are recognized; all input is merely collected. Input mode is left by entering a period (.) alone at the beginning of a line.

The ProLine version of *ed* does not support regular expression parsing, other than simple /text/ and ?text? syntaxes for bi-directional pattern searching. A null regular expression, // or ??, is equivalent to the last one used.

To understand addressing in *ed* , it is necessary to know that there is a current line at all times. Generally speaking, the current line is the last line affected by a command; the exact effect on the current line is discussed under the description of each command. Addresses are constructed as follows:

1. The character . addresses the current line.

2. The character $ addresses the last line of the buffer.

3. A decimal number n addresses the n-th line of the buffer.

4. A regular expression enclosed by slashes (/) addresses the first line found by searching forward from the line following the current line toward the end of the buffer and stopping at the first line containing a string matching the regular expression. If necessary, the search wraps around to the beginning of the buffer and continues up to and including the current line, so that the entire buffer is searched.

5. A regular expression enclosed in question marks (?) addresses the first line found by searching backward from the line preceding the current line toward the beginning of the buffer and stopping at the first line containing a string

matching the regular expression. If necessary, the search wraps around to the end of the buffer and continues up to and including the current line. Also see the last paragraph before Files below.

6. An address followed by a plus sign (+) or a minus sign (-) followed by a decimal number specifies that address plus or minus the indicated number of lines. The plus sign may be omitted.

7. If an address begins with + or -, the addition or subtraction is taken with respect to the current line; e.g, -5 is understood to mean .-5.

8. If an address ends with + or -, then 1 is added to or subtracted from the address, respectively. As a consequence of this rule and of rule 7 immediately above, the address - refers to the line preceding the current line. Moreover, trailing + and - characters have a cumulative effect, so -- refers to the current line less 2.

9. For convenience, a comma (,) stands for the address pair 1,$, while a semicolon (;) stands for the pair .,$.

Commands may require zero, one, or two addresses. Commands that require no addresses regard the presence of an address as an error. Commands that accept one or two addresses assume default addresses when an insufficient number of addresses is given; if more addresses are given than such a command requires, the last address(es) are used.

Typically, addresses are separated from each other by a comma (,). They may also be separated by a semicolon (;). In the latter case, the current line (.) is set to the first address, and only then is the second address calculated. This feature can be used to determine the starting line for forward and backward searches (see rules 4 and 5 above). The second address of any two-address sequence must correspond to a line that follows, in the buffer, the line corresponding to the first address.

In the following list of *ed* commands, the default addresses are shown in parentheses. The parentheses are not part of the address; they show that the given addresses are the default.

It is generally illegal for more than one command to appear on a line. However, any command (except e, f, r, or w) may be suffixed by p or by l, in which case the current line is either printed or listed, respectively, as discussed below under the p and l commands.

(.)a
<text>
.

The append command reads the given text and appends it after the addressed line; dot is left at the last inserted line, or, if there were no inserted lines, at the addressed line. Address 0 is legal for this command: it causes the ‘‘appended’’ text to be placed at the beginning of the buffer. This is equivalent to the 1i command.

(.)c
<text>
.

The change command deletes the addressed lines, then accepts input text that replaces these lines; dot is left at the last line input, or, if there were none, at the first line that was not deleted.

(.,.)d

The delete command deletes the addressed lines from the buffer. The line after the last line deleted becomes the current line; if the lines deleted were originally at the end of the buffer, the new last line becomes the current line.

e file

The edit command causes the entire contents of the buffer to be deleted, and then the named file to be read in; dot is set to the last line of the buffer. If no filename is given, the currently remembered filename, if any, is used (see the f command). The number of characters read is typed; file is remembered for possible use as a default filename in subsequent e, r, and w commands.

E file

The Edit command is like e, except the editor does not check to see if any changes have been made to the buffer since the last w command.

f file

If file is given, the filename command changes the currently remembered filename to file; otherwise, it prints the currently remembered filename.

h

The help command gives a short error message that explains the reason for the most recent ? diagnostic.

H

The Help command causes *ed* to enter a mode in which error messages are printed for all subsequent ? diagnostics. It will also explain the previous diagnostic if there was one. The H command alternately turns this mode on and off; it is initially off.

(.)i
<text>
.

The insert command inserts the given text before the addressed line; dot is left at the last inserted line, or if there were no inserted lines, at the addressed line. This command differs from the a command only in the placement of the input text. Address 0 is not legal for this command.

(.,.+1)j

The join command joins contiguous lines by removing the appropriate newline characters. If only one address is given, this command does nothing.

(.,.)l

The list command prints the addressed lines in an unambiguous way: a few nonprinting characters (e.g., tab, backspace) are represented by mnemonic overstrikes, all other nonprinting characters are printed in octal, and long lines are folded. An l command may be appended to any command other than e, f, r, or w.

(.,.)ma

The move command repositions the addressed line(s) after the line addressed by a. Address 0 is legal for a and causes the addressed line(s) to be moved to the beginning of the file; it is an error if address a falls within the range of moved lines; dot is left at the last line moved.

(.,.)n

The number command prints the addressed lines, preceding each line by its line number and a tab character; dot is left at the last line printed. The n command may be appended to any command other than e, f, r, or w.

(.,.)p

The print command prints the addressed lines; dot is left at the last line printed. The p command may be appended to any command other than e, f, r, or w; for example, dp deletes the current line and prints the new current line.

(*)P prompt

The editor will prompt with a * for all subsequent commands. If the optional prompt is included it becomes the new prompt string. P alone alternately turns this mode on and off; it is initially on.

q

The quit command causes *ed* to exit. No automatic write of a file is done.

Q

The editor exits without checking if changes have been made in the buffer since the last w command.

r file

The read command reads in the given file after last line in the buffer. If no filename is given, the currently remembered filename, if any, is used (see e and f commands). The currently remembered filename is not changed unless file is the very first filename mentioned since *ed* was invoked. If the read is successful, the number of characters read is typed; dot is set to the last line read in.

(.,.)s/regular-expression/replacement/

(.,.)s/regular-expression/replacement/g

The substitute command searches each addressed line for an occurrence of the specified regular expression. In each line in which a match is found, all (nonoverlapped) matched strings are replaced by the *replacement* if the global replacement indicator g appears after the command. If the global indicator does not appear, only the first occurrence of the matched string is replaced. It is an error for the substitution to fail on all addressed lines. Any character other than space or newline may be used instead of / to delimit the regular expression and the replacement; dot is left at the last line on which a substitution occurred.

(.,.)ta

This command acts just like the m command, except that a copy of the addressed lines is placed after address a (which may be 0); dot is left at the last line of the copy.

(1,$)w file

The write command writes the addressed lines into the named file. If the file does not exist, it is created. The currently remembered filename is not changed unless file is the very first filename mentioned since *ed* was invoked. If no filename is given, the currently remembered filename, if any, is used (see e and f commands); dot is unchanged. If the command is successful, the number of characters written is displayed.

($)=

The line number of the addressed line is typed; dot is unchanged by this command.

(.+1)

An address alone on a line causes the addressed line to be printed. A RETURN alone on a line is equivalent to .+1p. This is useful for stepping forward through the editing buffer a line at a time.

If an interrupt signal is sent, *ed* prints a question mark (?) and returns to its command level.

Some size limitations: 255 characters per line, 255 characters per global command list, 64 characters per filename. The limit on the number of lines depends on the amount of memory.

Files that contain characters not in the ASCII set (bit 8 on), cannot be edited by *ed*. Editable file types are TXT and CMD. TXT files cannot be the random-access text files.

Diagnostics

?	Command errors
? file	An inaccessible file

Use the help and Help commands for detailed explanations.

If changes have been made in the buffer since the last w command that wrote the entire buffer, *ed* warns the user if an attempt is made to destroy *ed*'s buffer via the e or q commands: it prints ? and allows you to continue editing. A second e or q command at this point will take effect.

See Also

edit(C), grep(C), csh(C), setenv(C), vedit(C)

Name

edit - Text editor

Syntax

edit *file*

Description

Edit is the standard ProLine line-oriented text editor. It is used to make changes to a text file, or create new files. The editor can hold roughly 300 lines of text. *Edit* recognizes the following commands.

Add	- Append text to end of buffer
Change <lines>	- Change text segments (local/global)
Delete <lines>	- Delete lines from buffer
Find <pattern>	- Search for a pattern of text
Insert at <line>	- Insert lines at specified location
List <lines>	- List buffer line(s) with numbers
Print <lines>	- Print line(s) without numbers
Quit	- Quit editing and save changes
Read	- Read a file into editor from user area
Wordwrap	- Toggle word wrapping ON or OFF
eXit	- Exit editor, cancel all changes
?	- Display a help file

<lines> is an argument which may be any of the following:

All	- All lines, same as pressing RETURN
First	- First line in buffer
Last	- Last line in buffer
.	- (period) Last line accessed

Or <lines> may be a numeric range as shown in these examples:

3	- Line 3
-3	- All lines from the first to 3
5-183	- All lines from 5 to 183
22-	- All lines from 22 to the last

When done editing, use Quit to write changes to disk and leave the editor. You can exit and cancel any changes by using eXit.

Editing a file does not change its file type. Only sequential text files can be edited.

See Also

ed(C), vedit(C)

Name

err - Describe error codes

Syntax

err [*number*]

Description

When an error occurs, a brief message is usually displayed describing the problem. However, some programs will just report an error code number. You can find out what an error code means by invoking the *err* shell script followed by a code *number*.

If you want to see the entire list of error codes and their descriptions, enter *err* without arguments.

Error reports with code numbers are usually accompanied by a line number reference (e.g. ‘‘Error #10 at 170’’). Supply only the error code (e.g. ‘‘10’’) but not the line number.

Files

$/etc/help/errors - database of error codes.

Name

find - Find files

Syntax

find [*options*] [*pathname*...] [>output]

Description

Find recursively descends the directory hierarchy for one or more *pathnames* given, seeking all files that match specified search options. If a *pathname* is ‘/’, the root prefix is assumed.

Without options, *find* lists all the files in the current working directory and every directory found within.

A number of useful examples are given at the end of this manual entry.

Options

-a	Consider all files when reading directories. This includes visible and invisible files. By default, only visible files are considered.
-c	Report a count of matching files when *find* is done.
-d levels	Search only through a specified number of directory *levels*. The default is 16 levels deep.
-f file	Use information obtained from *file* for making comparisons with searched files. Comparison is made for all search selectors except for file names (-N). Example: `find -f foo -T =` This finds all files within the current directory tree that match the file type for the file ‘‘foo’’. Using ‘=’ with a selector tells *find* to use the appropriate information from the file specified with -f. (Selectors are discussed below).
-i	Consider only invisible (hidden) files when reading directories.
-p pattern	Specify *pattern* when printing matched files. The contents of *pattern* are printed for each match, and imbedded ‘%’ characters are used to substitute special information: % - displays nothing %f - fully qualified path to file %n - name of file %p - parent path to file %s - size of file in blocks %x - partial path to file excluding source prefix

Combinations of these options can be included in the *pattern* (e.g., the pattern ‘‘%n found in %p’’ displays ‘‘bar found in /a/usr/foo/’’ when the complete pathname is ‘‘/a/usr/foo/bar’’).

By default, *find* displays the path to each file.

-q matches	Quit after the specified *matches* are found. The default is to never quit until all paths have been searched.
-t	Include the time (hh:mm) whenever a date match is made. By default, only the date (mm/dd/yy) is used in searches.
-v	Displays progress information on each directory being searched.

Selectors

The following selectors are used to specify matching files:

-A attrib Match files that have attributes as specified by these characters:

d = Destroy (file can be removed)
n = Rename (file can be renamed)
b = Backup (file needs a backup)
i = Invisible (file is hidden)
w = Write (file can be written to)
r = Read (file can be read)

-N pattern	Match files by a name *pattern*. The the *pattern* must be in lowercase, and may contain ‘*’ to match wildcard portions. Thus, ‘‘-N *baz*’’ finds all file names containing ‘‘baz’’.
-T type	Match files by *type* (a decimal value from 0 to 255).
-X auxtype	Match files by *auxtype* (a decimal value from 0 to 65535).
-S blocks	Match files by size in 512-byte *blocks*.
-B blocks	Match files bigger than the size given in *blocks*.
-L blocks	Match files smaller than the size given in *blocks*.

-M date Match files modified on *date*. *date* is a string in one of the forms listed:

```
mm/dd/yy
"mm/dd/yy hh:mm"
"mm/dd/yy hh:mm AM"
"mm/dd/yy hh:mm PM"
```

representing the month (1-12), day (1-31), year (0-99), hour (0-23), and minute (0-59). The string must be quoted if it contains a space. A period (.) indicates the current date and time.

If the time is omitted, midnight is assumed. If the time is significant in date comparisons, include the -t option as described above.

-O date	Match files older than *date*.
-Y date	Match files younger (newer) than *date*.

An equal sign (=) may be the argument to a selector if the -f option is used. See -f above for more details.

Negating Matches

Prefixing a selector with ‘!’ inverts (or negates) the match result. That is, ‘‘-!N’’ means to match all files that do not match a particular name pattern, thus ‘‘-!N *baz*’’ finds all files that do not contain the pattern ‘‘baz’’ in their names.

Composite Selectors

Multiple search selectors can be combined. As long as each selector matches, *find* continues to apply selection criteria to the current file. When all selectors have been processed successfully for a file, it is considered to be a match.

Output Redirection

If the very last argument given begins with ‘>’, *find* redirects all of its output, except verbose progress reports, into the filename which follows. This is excellent for creating scripts containing commands which act on all the matched files.

Examples

To find all executable programs younger (newer) than $/bin/man:

```
find -f $/bin/man -Y = $path
```

To build and run a script that removes all files beginning with ‘‘finder.’’:

```
find -N finder.* -p "echo found %n in %p; rm %f" / >killfndr
source killfndr
```

To find all files in $/usr that are not named ‘‘login’’, ‘‘signature’’, nor ‘‘mailrc’’, and are not directory files:

```
find -!N login -!N signature -!N mailrc -!T 15 $/usr
```

To report only the count of all invisible files in the current directory level:

```
find -i -d 1 -c -p %
```

To find all files that need to be backed up, showing their sizes:

```
find / -a -A b -p "%f (%s blocks)"
```

To find all files modified between January and February in 1994:

```
find / -a -Y "01/01/94" -O "02/01/94"
```

Author

Morgan Davis (mdavis@mdg.cts.com)

See Also

grep(C), setfile(C), version(C), whereis(C)

Name

grep - Find a pattern in a file

Syntax

grep [*options*] *pattern file*...

Description

Grep, abbreviation of Globally find Regular Expressions and Print, searches one or more *files* for lines matching a *pattern*. Normally, each line found is displayed. The following *options* are recognized:

-v	All lines but those matching are printed.
-c	Only a count of matching lines is printed.
-l	The names of files with matching lines are listed (once) separated by newlines.
-n	Each line is preceded by its relative line number in the file.
-h	Suppress display of filename headers.
-y	The case of letters is ignored in making comparisons (that is, upper and lower case are considered identical).

Example:

```
grep -y endif login
```

This displays all lines in the file *login* containing the word *endif*, ignoring alphabetic case.

Note

To search for a pattern consisting of words and spaces, the entire pattern must be enclosed in quotes:

```
grep "dream into action" clip
```

If the last argument in the command line starts with ‘>’, the output from *grep* is directed into the filename which follows:

```
grep -v mozart music.db >temp
```

This writes all lines in ‘‘music.db’’ not containing (-v) the pattern ‘‘mozart’’, into the file ‘‘temp’’.

Lines are limited to 255 characters; longer lines are truncated.

See Also

cat(C), tail(C), wc(C)

Name

help - Get help

Syntax

help

Description

Help is a shell script that can be customized to display helpful instructions while using the C-Shell (*csh*). Usually, this script simply aids in pointing the user in the direction of the online help system provided by the *man* command.

Files

$/etc/help/csh - the help file,
$/etc/help/whatis - a summarization of commands.

See Also

man(CT), manps(CT), whatis(CT)

Name

hist - System history

Syntax

hist *year*

Description

Hist, a shell script, displays a summary of changes made to the system software in the course of a given year. The *year* is a value minus any century information (e.g. ‘‘92’’, not ‘‘1992’’). Example:

```
hist 92
```

In this example, *hist* displays the system changes file for the year 1992.

Files

$/pub/proline/history.* - system change history files.

Name

if - Conditional command execution

Syntax

if *expression* then [*command1* [else *command2*]]

Description

If tests the *expression*, and if it results in a true value, performs *command1* following the *then* keyword. If the *expression* is false, and an *else* keyword is given, *command2* is performed.

Multiple Lines

If statements can span more than one line in order to issue multiple commands. This is an example of a compound *if* statement:

```
if expression then
     command
     command
     ...
endif
```

The *endif* keyword denotes the end of a compound *if* statement. Here's an example using an *else* option:

```
if expression then
     command1
     ...
else
    command2
     ...
endif
```

Expressions

Typical expressions include:

x = y	Equality (== can be used)
x <> y	Inequality (!= can be used)
x > y	Greater
x < y	Smaller
x >= y	Greater or equal

x <= y	Less or equal
-f file	Existence of a file

The values for x and y in expressions can be any kind of argument, including variables. Multiple expressions can be included as well, with parenthesis used for precedence. To mix expressions logically, separate them with these two keywords or symbols: and (&&), or (||). To negate the result of an expression, precede it with the keyword *not* (also !). Example:

```
if $a = $b then echo TRUE else echo FALSE
```

Another example:

```
if $time[0] = "Sun" and -f workfile then
     echo It is Sunday AND the workfile exists!
else
     echo EITHER it is not Sunday
     echo OR the workfile does not exist.
endif
```

Nested Statements

Nested *if* statements can be used in a script, as shown:

```
if ( expr1 ) then
     if ( expr2 ) then
          echo "expr1 AND expr2 are true"
     endif
else
     echo "Either expr1 OR expr2 is false"
endif
```

Each *if* must have a matching *endif*.

See Also

csh(C), scripts(M)

Name

import, export - Text file conversion

Syntax

import *file1* [*file2*]
export *file1* *file2*

Description

Import is used to filter linefeeds after carriage returns from a text file. This is handy after uploading a text file containing unnecessary linefeeds after carriage returns from operating systems such as CP/M or MS-DOS.

Import reads *file1* and writes the filtered output to the optional *file2*. If *file2* is omitted, *import* just converts *file1*.

Export is used to append linefeeds after carriage returns just prior to downloading to an operating system which needs CR/LF newline sequences.

Note

Export is not as flexible as *import* in that you must specify *file2* when exporting *file1*.

Some transfer protocols include newline conversion.

See Also

rx(C), rz(C), sx(C), sz(C)

Name

inspect - Inspect contents of files

Syntax

inspect [*options*] *file*

Description

Inspect describes the contents of *file*. If *file* is determined to be a NuFX (‘‘SHK’’), Compact Pro (‘‘CPT’’) or StuffIt (‘‘SIT’’) archive, its enclosed files, folders and disk images are listed. If *file* is determined to be a CompuServe GIF(tm) format graphic, information about its size and color depth is provided.

Inspect takes two options:

-b displays Binary II or MacBinary information if present.

-i identifies the file as a particular type, but does not list its contents. If the last argument given to *list* begins with ‘>’, it redirects output to the filename that follows.

Notes

Full support for self-extracting CPT and SIT archives is provided, but support for self-extracting SHK archives created by GS ShrinkIt is not.

Primitive support for ‘‘MOD’’ format music files is provided in that files beginning with ‘‘MOD.’’ or ending with ‘‘.MOD’’ list their internal music score names.

The ProDOS 8 filetype lookup table has been placed at the end of the program to facilitate modification or expansion by any BASIC programmer.

Inspect uses 16-bit CRC routines placed in the public domain by Jim Ferr.

Diagnostics

‘‘Could not load OMM module CRC’’ -- the support module $/sys/modules/CRC could not be found and loaded into memory.

‘‘NuFX master header damaged’’ -- The file is determined to be a SHK archive, but its master header is not in the proper format.

‘‘NuFX local header damaged’’ -- The file is determined to be a SHK archive, but one or more of its entries are not in the proper format.

‘‘StuffIt archive header damaged’’ -- The file is determined to be a SIT archive, but its master header is not in the proper format.

Files

$/sys/modules/CRC - support module with CRC routines

See Also

unpar(C)

Author

Jon C. Thomason (jonct@pro-applepi)

Name

it - InteleTerm Pro terminal program

Syntax

it [*name*]

Description

InteleTerm Pro, *it*, is a full-featured terminal program for local console use. It contains many commands and features to simplify your online session. It also has the ability to run special scripts containing commands to automate a login procedure and other online tasks.

Invoking *it* brings up a help screen showing the commands you can enter. If you have a compiled script you wish to invoke, you can include its *name* as an argument to have it invoked upon startup. Example:

```
it bix
```

This would launch *it*, which would then attempt to run the ‘‘bix’’ script. (See the manual entry for *itc*, the InteleTerm Compiler, for more details on *it* scripts).

While offline, *it* displays a command prompt at the top of the screen. Simply enter one of the command letters to invoke a command. While online, you can invoke those same commands from terminal mode by holding down the Apple (or Command) key while pressing a command letter. You can return to terminal mode by pressing the Space Bar at the Command prompt.

You are free to leave *it* while online to use the shell. To return to your online session, simply invoke *it* with no arguments.

Learning to use *it* is best done by use.

See Also

itc(C), sm(C)

Name

itc - InteleTerm Pro script compiler

Syntax

itc [*name*]

Description

Itc is a script compiler that generates scripts for use with *it*, the InteleTerm Pro terminal program.

Without arguments, *itc* asks for the name of a script to compile. You can also the script *name* in the command line by including its name only (not the full path) as an argument to *itc*. Leave off the .src extension as *itc* adds it for you. Example:

```
        itc bix
```

This compiles the ‘‘bix.src’’ script, writing the compiled output to ‘‘bix’’. You can also use the [C]ompile command from the [S]cript menu within *it*.

Creating and Editing Scripts

It scripts can be created with your text editor. From the [S]cript menu in *it*, choose [N]ew to create a new script, or [E]dit to edit an existing one. Again, leave off the ‘‘.src’’ extension when referencing your script.

The format of a script file follows these rules:

1. Each instruction is on a line by itself
2. Labels are declared on lines by themselves and a colon (:) follows the label name.
3. When labels are referenced, the colon is not used.
4. Indenting is optional, but aids in keeping your script readable.
5. Quoted text arguments can have control characters imbedded in them by entering a caret (^) followed by the letter (upper or lowercase) of the corresponding control character. For example, to put a carriage return at the end of a PRINT instruction, you would use:

   ```
   PRINT "Testing^M"
   ```

6. Quoted text arguments can include characters like quotes, carets, and backslash by prefixing them with the backslash (\).
7. Text arguments containing spaces must be put between quotation marks, otherwise the program will only recognize the first word in a series of words.

8. A space character must separate a command from its argument(s).

9. Comments can be placed on lines by themselves following any of these comment characters: pound (#), semi-colon (;), apostrophe (’), asterisk (*).

Scripting Language

The script language consists of these commands:

do label	Causes script execution to begin executing instructions at *label*. When a return instruction is encountered, program flow resumes with the next instruction following the do command.
term	Temporarily pauses script execution and puts you into terminal mode. When you exit terminal mode, script execution continues.
goto label	Diverts script execution to a certain label. Unlike do, a return will not return to the instruction after the goto.
clear	Clears the screen and puts the cursor in the upper left corner.
print "text"	Displays text on the screen, but is not sent to the modem. Add ‘‘^M’’ at the end of the text to issue a carriage return.
speed bps	speed will set the bits per second rate for use with *it*. Values for *bps* are 300, 1200, 2400, 9600, 19200, 38400, and 57600.
echo mode	Sets up local or remote echo modes for terminal mode. Arguments for *mode* are on and off. Echo on is half duplex (local echo). Echo off is full duplex (remote echo).
dial number	Dials a phone number.
connect	Attempts to connect with a host computer after dialing. Use the IF statement after executing connect to determine if connection was successfully established.
return	Causes script execution to return to the line following the corresponding do instruction.
wait "text"	Halts script execution until ‘‘text’’ has been received. This is used for handshaking on incoming data. Use the if instruction to determine if pattern was found.
if condition [then goto] label	IF is used to test the most recent function performed by the script, such as attempting to connect with a host computer, sending or receiving a file, or handshaking on incoming text. Example conditions for use with IF are:

```
if found goto ...    if ok goto ...
if connect goto ...  if good goto ...
if failed goto ...   if bad goto ...
if true goto ...     if success goto ...
if false goto ...    if no goto ...
if yes goto ...
```

Place not or no in front of a condition to negate it, such as ‘‘if not found goto label’’. The words ‘‘then’’ and ‘‘goto’’ are optional. If the condition is true, the script branches to the label referenced at the very end of the if instruction.

send "text" Works like print, except sends text to the modem and not to the screen. If you need to send a carriage return, use ‘‘^M’’ (caret followed by the letter M).

hangup Disconnects the modem with the host, and hangs up the phone.

sendfile file [delay [char]]
Send the text in *file*, line by line. Output is paused after each line for *delay* seconds if included. The next line is not sent until *char* is received if the optional *char* argument is included. Example:

```
sendfile bletch.txt 2 :
```

This sends the file ‘‘bletch.txt’’ line by line, with an interline delay of 2, and waits for a colon from the host before each line is sent.

break Sends a modem break tone.

sleep seconds Causes script execution to pause for the *seconds* specified, then execution resumes.

exec command Executes the system command line in *command*. This can be used to run any program on the system, such as external protocols, or a C-Shell that invokes a shell script. Examples:

```
     exec "csh cd /profile/mail; rm tempfile"
     exec "sz filename"
     exec "csh source batch.script"
```

Commands with spaces must be enclosed in quotation marks.

When the command is complete, control returns to *it* and the script continues. ‘‘If’’ statements can be used to test the success of the exec command.

run script Lets you run a different script file from within a script.

end	Causes a script to stop running. If you’re online, you’re taken right to terminal mode.
exit	Causes the script to stop running and forces *it* to quit.
printer mode	Specifies whether any output which is shown inside the terminal mode window will be sent to the printer or not. Values for *mode* are on and off. Printer on sends output to the printer, while printer off does not.
emulate terminal	Initializes terminal emulation for the named terminal template file.

Function Key Labels

Labels that begin with the name ‘‘fkey’’, followed by a digit from 0 to 9 (e.g., fkey5:, fkey8:, etc.), denote entry points in loaded scripts when the Command key and a digit are pressed in terminal mode. Issuing a function key is like choosing the Goto command from the Scripts menu in *it* and selecting a label. As soon as the script ends, control returns to terminal mode. This makes it possible to create ten key-triggered functions in each script.

Files

$home/src - directory containing scripts.

See Also

it(C)

Name

log - Display log files

Syntax

log [*option*]

Description

Log displays the various contents of the system's log files. *Log*, without any options, displays actual logins only. Other events can be shown selectively by specifying the following options:

-a	Show entire log, same as cat $spool/logs/syslog.
-c	Lists all *cron* and *kybd* tasks that were issued.
-g	Display log of guest callers only.
-m	Display the mdss log, same as cat $spool/logs/mdsslog.
-M	Lists only the major entries in the mdss log.
-mp	Prints only mdss POLL attempt entries.
-mm	Prints only MDSS login entries.
-n	Lists the newsgroup processing log, same as cat $spool/logs/newslog.
-s	Display the server's log, same as cat $spool/logs/servlog.
-u	Display the new user log, same as cat $spool/logs/adduser.log.

Example:

```
log -c
```

shows the log of *cron* events and *local* commands that have occurred.

Files

$spool/logs/adduser.log - new user log file,
$spool/logs/mdsslog - mail delivery subsystem log file,
$spool/logs/newslog - newsgroup log file,
$spool/logs/servlog - file server log file,
$spool/logs/syslog - system log file.

Note

Because *log* is a shell script, multiple options cannot be selected.

See Also

calls(C), uutraf(NET)

Name

login - Sign on

Syntax

login

Description

The *login* program is used when a user initially signs on, or it may be used at any time to change from one user to another without having to disconnect.

After a normal system startup sequence, the *login* program is executed. Its job is to answer the phone when remote callers dial the system, prompt them for a login code and password, and then allow them into the system if the codes are valid. While waiting for a call, it displays the current time on the screen. *Login* is also responsible for handling scheduled *cron* tasks.

Commands While Waiting

While waiting for a call, the console operator can instruct *login* to process a command by pressing the '!' key. *Login* reads a command line from the operator and executes the first command passing forward any arguments. For example, entering ''!ctime'' causes *login* to run the *ctime* program as a process. When *ctime* exits, control returns to *login*.

Other keys that *login* recognizes while waiting are:

RETURN	Allows the operator to log into the system locally
SPACE	Toggles the screen saver on and off
ESCAPE	Exit to BASIC

In addition, certain keys can be configured to invoke macro commands to perform a variety of functions without actually having to login. See the Resources section of this manual entry for more details.

Task Scheduling

Login includes an internal task manager called *cron* which runs once a minute. *Cron* scans the file $/etc/crontab for tasks to perform at certain times and executes them in a way similar to the '!' command for local console commands. (See *cron* for more details).

Any processes that *login* invokes are logged with the date and time of execution, including elapsed time. Local console processes are labeled ''local'' while *cron* processes are labeled ''cron'' in $spool/logs/syslog.

Intercepting Phone Calls

Unless otherwise instructed (see *Resource File* below), *login* answers a call on the first ring and attempts to connect. The connection speed is verified with a required rate, if specified. If the connection is slower than desired, the $/etc/lowbps file is displayed, and *login* disconnects to wait for another call.

After *login* answers a ringing phone line and successfully connects with another computer, it displays the file $/etc/herald. This file contains the name and location of the host system. The caller is prompted to enter a valid login name.

After a valid login name is given, *login* scans $/etc/passwd for a matching entry. Once found, the information for that account is read in. If the account is a non-root user, and the file $/etc/nologin exists, *login* prints its contents on the user's terminal and disconnects. This stops users from logging in when the system is about to go down, or when it is closed to user traffic.

If the interpreter entry for the login is a quoted message, it is displayed, and the caller is prompted to enter a login name again. (e.g., a ‘‘help’’ login attempt might display a short message for signing on using a public account. Use *adduser* to create a ‘‘help’’ login which would display the quoted message given as the interpreter).

Next, *login* prompts the caller for a password, if appropriate. Echoing is turned off during the typing of the password, so it will not appear on any written record of the session. However, if the encrypted password in the entry deciphers to ‘‘none’’, password input is bypassed. This is usually the case for guest logins.

The user has three attempts to enter a valid login name and matching password. If the caller fails, the file $/etc/badlogin is displayed, and *login* disconnects. Badlogin contains information on how to contact the system administrator for assistance, or may offer instructions for using a public account.

If all is well, the user is shown the previous sign on date, the message of the day file ($/etc/motd), and if a guest, the welcome file ($/etc/welcome). *Login* then executes a command interpreter according to the user's password file entry. The current working directory will be set to the user's home directory.

Screen Saver

While *login* is waiting for a call or a command, it blacks out the local console display after a period of inactivity. This prevents images from ‘‘burning’’ into the screen. If a call comes in, a key is pressed, or a cron task starts up, the screen is restored. The time is displayed at random positions while the screen is blank.

Resource File

Certain features of *login* can be adjusted by editing $/etc/rsrc/login.rsrc. The format of the file is described by the following sample entries in the order required (with comments added):

30	Seconds before the screen blanks. Zero seconds disables the screen saver.
<blank>	Normally left blank, this line may contain a command line that is executed whenever a caller logs out or a cron task completes. By default, no command line tells *login* to scan for any letters that need delivering, and if found it executes *sendmail*. Typically, a command line would start with *csh* and the name of a shell script that contains a list of commands to execute, including *sendmail*.
0	Speaker control. This feature allows you to control the modem's speaker via joystick position control. If this entry is 1, the direction of the joystick affects the modem's speaker output during connections. Positioned to the left, the speaker remains silent.
300	Lowest connect speed allowed. If a caller connects at a speed lower than this value, $/etc/lowbps is displayed, and *login* disconnects.
0	Console availability hold time. This is a feature for Apple IIGS hosts only. This feature allows the administrator to lock the Caps Lock key while the system is in use. When the system becomes available, a series of beeps (the number determined by this entry's value) is emitted, holding the system until there is a keyboard response. Unlocking the Caps Lock key, or setting this entry to 0, disables this feature.
!cmpqs	Macro command keys. Each letter in this line corresponds to subsequent lines that expand the macro key into commands. That is, the first line following this one is for the '!' character. The second is for the 'c' character, and so on. The macro lines must follow in the order that the characters are given. Up to 26 macros may be defined. Shown here are the defaults with comments in parenthesis (omit the comments):

```
            (for "!" -- usually nothing)
csh -c      (for "c")
mail        (for "m")
poll        (for "p")
boot -q     (for "q")
scan -l     (for "s")
```

Pressing 'q' at the Waiting prompt expands into the ‘‘boot -q’’ command. This is because 'q' is the fifth character in the first line of macro keys, and ‘‘boot -q’’ is the fifth line following. You can change or edit the command line, then execute it by pressing RETURN. You can, of course, add, delete, or change any these macros.

3	Console cancel key (ASCII). The default console cancel key is Control-C (ASCII 3). This can be changed to any other character from ASCII 0 to 127. This cancel key is used whenever a macro is invoked.

0	No answer flag. If set to 1, causes *login* to ignore any incoming calls. It simply prevents the system from answering, allowing the modem to remain operational for outgoing calls only.
0	Command line time-out. This value (in seconds) sets the duration of inactivity at the Waiting command line prompt. If zero, no inactivity checking is performed.
<blank>	This line, normally blank, contains a comma-separated list of account names that triggers a unique alert sequence. When a user signs on whose login name is in this line, a series of beeps is generated at the console. Then a second set of beeps follow, unique to that user. You can distinguish certain users from others without having to look at the console. This is useful for alerting you to troublesome callers.

Files

$/etc/rsrc/login.rsrc - Login resources,
$/etc/adm - Accounting,
$tmpdir/utmp - Accounting,
$/etc/crontab - Task table,
$/etc/herald - Login herald,
$/etc/motd - Message of the day,
$/etc/passwd - Password file,
$/etc/lowbps - Message indicating connect speed requirements,
$/etc/nologin - Stops non-root user logins,
$/etc/badlogin - Shown after three failed login attempts,
$/etc/welcome - guest welcome,
$/adm/*/environs - User’s terminal environment settings,

Diagnostics

‘‘Login incorrect,’’ if the name or the password is bad.
‘‘Account overdrawn,’’ when a caller has used up his monthly time allotment.
’’File or directory not found,’’ when the specified user’s directory or interpreter to launch cannot be found.

See Also

adduser(ADM), cron(ADM), csh(C), passwd(C), plush(C), setenv(C)

Name
logout - Terminate connection

Syntax
logout

Description
Logout breaks the connection with a caller. It cancels all processes and exits to the *login* program, disconnecting the caller.

See Also
boot(ADM), csh(C), login(C)

Name

lpr - Send files to the printer

Syntax

lpr [options] *file*...

Description

Lpr prints files to a locally connected printer. One or more files may be printed at a time. The options are:

-i chars	Send initialization characters to your printer before printing a *file* (default is no initialization).
-s slot	Sets the printer slot (default is slot 1).
-f	Disables the formfeed (Control-L) sent after each file (default is enabled).

Lpr works with any type of printer interface.

Also See

cat(C), csh(C)

Name

ls - List directory contents

Syntax

ls [*options*] *directory...*

Description

Ls lists the files in one or more *directories*. If no *directory* is given, files in the current working directory are shown. Without options, *ls* displays the filenames in a directory in five columns, sorted alphabetically.

Ls takes a variety of options:

-a displays all files in the directory, including the ‘‘hidden’’ files.

-l displays a ‘‘long’’ directory, showing file information for each filename.

Example:

```
total 4 for 821 bytes
-rw- txt     258 13-aug-85 23:11  dead.letter
drw- dir     512 19-aug-85 00:42  letters
-rw- txt      33 14-aug-85 20:46  patch
-rwx cmd      18 12-aug-85 16:43  login
```

The first line shows the number of files within the directory and the amount of space (in bytes) occupied by them. For each line of file information:

```
 -rwx cmd      18 12-aug-85 16:43  login
    |   |        |       |            |
    |   |        |       |           name of file
    |   |        |       |
    |   |        |      date file was last modified
    |   |        |
    |   |       size of file (in bytes)
    |   |
    |  file type
    |
 d--- indicates a directory
 -r-- file can be read by owner
 --w- file can be written to by owner
 ---x file can be executed by owner
```

-n reports the number of files in the specified directory.

-s causes a long directory to express all size values in 512 byte blocks, instead of bytes.

-F marks subdirectory files with a trailing ‘/’, and executable files with a trailing ‘*’.

-p displays a directory in standard ProDOS format (full display, all information). This option supersedes all others.

-i shows only invisible files (not available with the -l option).

In all forms except -p, *ls* supports output redirection to files. Example:

```
ls /heartbeat/city >file.list
```

This redirects the columnar output from *ls* on the /heartbeat/city directory into the file ‘‘file.list’’.

Diagnostics

‘‘(more files exist than shown)’’ -- Too many files exist in the directory and all their names could not be read into memory. If the ‘-p’ option is used, there is no file count limitation.

See Also

dstat(C), setfile(C)

Name

mail - ProLine mail

Syntax

mail [-m mbox]
mail [-n]
mail [-s] user ...

Description

Mail is an interactive electronic mail management system that lets you send, receive, and store messages. Because of its interactive nature, *mail* is best learned by use. To invoke *mail*, use any of the following examples:

mail	Reads messages in your mailbox.
mail -m letters	Reads messages in the file ‘‘letters’’.
mail -n	This flag causes mail to display ‘‘No new mail’’ and exit if your mailbox has not been changed since the last time you read mail.
mail user	Sends mail to ‘‘user’’.
mail -s	*Mail* prompts you for the name of a user to send to.

The command levels in *mail* respond to single-key abbreviations for command names. They are expanded for you automatically. A list of available commands is displayed by typing a ‘‘?’’ at any prompt.

Upon entering *mail*, the mailbox will be scanned for all your messages. A report is then listed which shows various information about each message, including the sender’s name and the subject.

Reading Mail

mail recognizes the following interactive commands at the mail> prompt. To invoke a command, enter the first letter of its name:

[RETURN] / [SPACE]	Pressing the RETURN or SPACE keys advances you to the next message, and displays it. (Displays the first message if none have been read.)
Next	Marks the current message for deletion, then displays the next message. (Displays the first message if none have been read.)
Previous	Displays the previous message.
Type <msgs>	Displays the selected messages. The ‘‘current message’’ number will be set to the last message printed.

Again	Displays the current message.
Headers <msgs>	Prints a one-line header for the selected messages. The header contains: a '*' if that message is marked for deletion, the number of the message, the number of characters in the message, the sender's name, and the subject.
List	Lists headers for all the messages (same as Headers All).
Send	Sends a message. You are asked to fill in the ''To:'' field with the name of the recipient (enter nothing at all to cancel), and the subject of the message. You may then start typing the body of your message. (see *Sending Mail* below for more details)
Forward <msgs>	Forwards selected messages to others. You address the message as with Send above, and then enter some introductory comments of your own which will appear at the start of the message. Once you leave text entry mode, the forwarded message is appended into the outgoing message.
Reply to <msg>	Sends a reply to a message. *mail* fills in the ''To:'', and ''Subject:'' lines of your reply. You have the option of entering the addresses for additional recipients (if any) in order to send ''carbon copies'' (Cc:).
Delete <msgs>	Marks selected messages for deletion when quitting. Messages marked for deletion will have a '*' after their number in their header. Deletion marks may be removed with the Undelete command. Note: this only marks messages to be deleted later. Marked letters are still accessible until you Quit.
Undelete <msgs>	Removes the deletion marks for the selected messages.
Write <msgs>	Writes the selected messages to a file. If the target file exists, you are asked if the messages should overwrite it or be appended to it.
View	Views a file from disk.
Edit	Allows you to edit a file while working in *mail*.
$ Edit Vars	Invokes the editor on your ''mailrc'' file which defines variables that you may use with *mail*. (See *Variables* below for details.)
! Shell Command	Executes a command line you enter at the ''!'' prompt. Commands can be anything that can be executed in a normal command line shell, including invoking an interactive shell (*csh -c* for example). When the command is completed, you can continue to enter additional commands at the ''!'' prompt, or press Enter to return to the current prompt in *mail*.

Open	Opens a new mailbox, allowing you to close the current one. If the current mailbox has messages marked for deletion, you are asked to confirm their removal before the new mailbox is opened and processed.
Quit	Quits *mail*. You are asked to confirm the removal of any messages marked for deletion before quitting.
? (help)	Shows a list of all the available commands.

The <msgs> argument to some commands may be one of the following:

All	All of the messages in the current mailbox
Current	Last message typed (same as RETURN)
Next	Next message (current message + 1)
Previous	Previous message (current message - 1)
First	Message number 1 (the oldest message)
Last	The highest numbered (most recent) message

Or <msgs> may be a list of numbers, as in these examples:

2	message 2
11-15	messages 11 through 15 (inclusive)
-15	first message through message 15
3-	message 3 through the last message

Special Keys

All command levels in *mail* support the + and - keys which turn the --*More*-- prompt on and off, respectively. Additionally, the mail> and send> prompts allow you to type in a number (or numeric range) to display the specified messages from your mailbox. Using the ‘‘;’’ key at any command prompt will list the headers of all messages in your mailbox (same as the List command at the mail> prompt).

Mail Strategy

Here’s a strategy for reading and processing your mail efficiently using *mail*. Upon entering *mail*, examine the report of your messages to get an idea of the kind of mail you have, the sizes of various messages, and whether or not there are any which require immediate attention.

If you don’t have any urgent messages, you can read each in turn with the Next command. Once a message has been displayed, it is referred to as the ‘‘current’’ message. You can reply to the current message with the command ‘‘Reply to Current’’; this begins an outgoing message in response to the message you’ve just read.

Using Next automatically marks the current message for deletion, and then proceeds to display the next message. If you press RETURN or SPACE to advance to the next message, the status of the current message is not affected. You can, however, mark any message to be deleted later with the ‘‘Delete’’ command. You may also want to proceed to the next one. If you want to read a message out of order, use the ‘‘Type’’ command followed by the number of the message and a RETURN. Or, you can simply type in the message’s number (or a range) at any prompt, followed by a RETURN.

In general, it is easier to deal with each message one at a time, as you read it, although some people like to read all of their mail first, then go back and delete and/or store away some of their messages.

When you are ready to leave *mail*, use the ‘‘Quit’’ command. If you have marked any messages for deletion, you will be asked if you actually want them permanently removed. You may save a message marked for deletion at any time with the ‘‘Undelete’’ command.

Sending Mail

Mail lets you send a message to one or more users on your local site, as well as to users on distant sites on the network. To address a local message, you provide the user’s account name (not the user’s actual name). You can specify a group of names (each separated by a space) in order to send a single message to multiple users. Network mail addresses can be entered using the ‘‘user@site’’ or ‘‘site!user’’ syntaxes.

After entering the address of the message, you are asked to fill in the ‘‘Subject:’’ field. Then, you’ll be placed in text entry mode. Anything you type that does not start with your exit character (usually a period ‘‘.’’) is added to the body of your letter. Word wrapping occurs at column 75. To exit text entry, type the exit character at the beginning of a new line followed by RETURN.

The following interactive commands are recognized at the send> prompt, as well as from text entry mode by prefixing them with the exit character (i.e., ‘‘.s’’ is the same as typing ‘‘s’’ from the send> command prompt).

Add	Continues to add text at the end of your message.
List	Lists the header and body of your message.
Edit	Calls up the editor for use with your message. After saving your editing changes, you’ll be returned to the send> prompt for further commands.
Include <msgs>	Includes the bodies of the specified messages from your mailbox into the message, prefixing each line with the prefix character as defined in the $prefix variable (usually the > symbol).
Headers ... <msgs>	Same as Include above, but it includes the header and body portions of messages.
Forward <msgs>	Includes the header and body of the specified messages without formatting.

Get	Gets a file from disk and appends it to the end of the message.
Send	Sends the message to all the specified recipients. If any outgoing network addresses were specified, the file named *signature*, if it exists in your home directory, is appended to the end of the message.
Modify	Modifies the To, Subject, Cc, and Bcc fields in the header of your message.
To	Specifies the target recipient (To) addresses.
Reply-To	Specifies a Reply-To address for when the recipient of the letter generates a reply, the reply will be sent to the address in this field instead of the original author of the message.
Cc	Specifies ‘‘carbon copy’’ (Cc) addresses.
Bcc	Specifies ‘‘blind carbon copy’’ (Bcc) addresses. These are like Cc’s, but only the users in the ‘‘Bcc:’’ field will be aware that they received copies of your message. The recipients in the ‘‘To:’’ and ‘‘Cc:’’ fields will not be aware that additional copies were sent.
View	Views a file.
Write	Writes the body of your message to a file (with the option of to include the header).
$ Edit Vars	Allows you to edit or peruse your variables. (See *Variables* below for details.)
! Shell Command	Execute a command line. earlier).
Open	Opens a new mailbox.
Quit	Quits without sending the message.

Messages are not actually delivered until you log off the system or issue the *sendmail* command from the C-Shell.

Variables

Mail accepts variables as shortcuts for lengthy or hard to remember input. A variable is a keyword that begins with a dollar-sign ($), such as $prompt. When used in areas where text arguments are supplied, the values of these variables are inserted (except in text input mode and the ‘‘Subject:’’ field of outgoing messages). Variables can also be used to control and customize the behavior of *mail*.

Some variables are defined for you automatically:

/	Contains the system’s startup prefix. (Cannot be deleted)
addsig	If set to 1 (default), *mail* attaches your ‘‘signature’’ file to messages you send.
askcc	If set to 1 (default), you’re asked if you want to include all recipients in the carbon-copy field of outgoing messages.
clear	Controls the clearing of the screen at various points in *mail*. If set to 0, disables any screen clearing. If set to 1 (the default), *mail* will clear the screen upon entry and before any list, file, or message is displayed.
current	Keeps track of the current message between sessions. Delete this variable to disable this feature.
delmark	The character used to mark deleted messages, usually ‘‘*’’.
escape	If this variable is not defined or empty, your preferred editor will be invoked for message input. By setting the escape variable to any character, *mail* uses its built-in input mode for message composition. (You can still invoke your editor later). The built-in editor accepts one line of input at a time, until you enter a line that contains only the character stored in the escape variable, which lets you ‘‘escape’’ from input mode. Escape sequences may include one extra character which causes *mail* to exit input mode and act on the next character as if it were entered at the send> prompt. The default escape is ‘.’, causing *mail* to use the built-in input mode.
home	The path to your home directory. (Cannot be deleted)
lastime	This variable should not be changed. It is maintained by mail to keep track of the last time you read any mail. Its purpose is to hold the date and time (in a special decimal format) for use with the -n flag when *mail* is invoked.
mail	The path to your incoming mailbox. (Cannot be deleted)
nomail	Controls exiting from *mail* if the mailbox is empty. If set to 0, invoking *mail* with an empty mailbox leaves you at the mail> prompt. If set to 1, an empty mailbox causes mail to report, ‘‘No mail,’’ and then quits.
org	Holds the name of the user’s organization that is placed into the Organization: field of outgoing messages.
prompt	This character (or string) is displayed for each new line of input while in built-in text input mode. (See *escape*).
quote	This character (or string) is used to ‘‘quote’’ each line included from your mailbox into a reply.

replyto	Holds an address that is placed into the Reply-To: field of outgoing messages.
summary	Controls the display of the message summary after entering *mail*. If set to 0, no summary is displayed. If set to 1 (the default), a summary of all messages in your mailbox is shown. If set to 2, a summary of only the current message to the last is shown.
wrtdel	If set to 1 causes messages written to a file to be marked for deletion. (default is 0)

The dollar-sign ($) introduces each variable used as input. If you wanted to write a message to a file, you could enter:

```
mail> Write Current
to file: $home/tempfile
```

mail sees $home and substitutes it for the path to your home directory, saving you a few keystrokes.

Another example: You regularly correspond with a group of writers. Typing in all the names is a tedious task. Instead, you create a variable to make life much simpler. Let's say you've created your own variable called ''writers'' in your *mailrc* file:

```
writers=brock ryan dang rlouv davew garyw@site.com
```

To use it, follow this scenario:

```
mail> Send
To: $writers
```

Mail replaces your variable with its value, which actually ends up addressing the message like so:

```
To: brock ryan dang rlouv davew garyw@site.com
```

Variables can be used to substitute any part of a pathname, address, or filename. They're not case-sensitive, and can reference other variables.

Files

$home/mailrc - variable assignments,
$/usr/*/signature - signature file,
$/adm/*/mailstop - prevents sending offsite mail.

See Also

sendmail(C), rcp(NET)

Author

Morgan Davis

Name

mcinews - Poll MCI Mail for business news headlines

Syntax

mcinews [-d]

Description

Without arguments, *mcinews* displays two business and financial news headlines which are updated every four hours by the MCI Mail information service.

With the -d argument, *mcinews* reads the resource file $/etc/rsrc/mcinews.rsrc for information on how to dial MCI Mail, login, and download the daily business news headlines.

The resource file contains four lines of information:

```
1-800-555-1212
mdavis//arglebop
2400
mdavis
pub/local/mcinews
```

These lines correspond to:

Phone	The phone number to the MCI Mail service.
Acct Info	Login name and password (separated by "//")
Speed	The baud rate at which the connection should be made.
Local Acct	Login name of the user on ProLine who owns that MCI Mail account (a future version of *mcinews* may be able to capture that person’s mail on MCI and forward it to the appropriate ProLine mailbox).
Pathname	Pathname to the file that contains the headlines. If it doesn’t start with a slash (/), the ProLine system prefix is assumed.

Files

$/etc/rsrc/mcinews.rsrc - resource file.

Name

mkdir - Make a directory

Syntax

mkdir *directory*...

Description

Mkdir is used to make one or more *directories*. Directories, like filing folders, contain files (usually related in some way) as well as other directories. For example, suppose you wish to create a directory within your home directory to store miscellaneous tidbits (as well as any simple desultory philippics) that might come your way. To do so with the name ‘‘stuff’’, you would use:

```
mkdir stuff
```

To access files in your new directory, you would precede the filenames with ‘‘stuff/’’ (the ‘/’ being the delimiter). For example, to view a file in ‘‘stuff’’ from your home directory, type:

```
cat stuff/filename
```

See Also

cd(C), dstat(C), ls(C), rmdir(C)

Name

more - Browse text files

Syntax

more *file*...

Description

More is used to show the contents of a *file*, one screen at a time. Any type of file can be displayed, but ASCII (TXT) and script (CMD) files will produce the only meaningful results. For example,

```
more login
```

displays the contents of your *login* script. More than one file can be shown by giving *more* a file list, such as:

```
more mailrc resume login
```

When at the -- *More (xx%)* -- prompt, the following commands are available (*i* denotes a number):

i SPACE	Display another screenful, or *i* more lines if *i* is specified.
i RETURN	Display another line, or *i* more lines, if *i* is specified.
i ^D	(Ctrl-D) Display 11 more lines. If *i* is given, the scroll size is set to *i*.
i d	Same as ^D (Ctrl-D)
i z	Same as SPACE, except that *i*, if present, becomes the new default number of lines per screenful.
i s	Skip *i* lines and then display a screenful.
i ^B	(Ctrl-B) Skip back *i* screenfuls and then display a screenful.
i b	Same as ^B (Ctrl-B)
e	Edit the current file using the editor selected in setenv. When done editing, return back to *more*.
=	Display the current line number.
i :n	Skip forward *i* files given in the command line, or to the last filename in the list if *i* is out of range.
i :p	Skip backward *i* files given in the command line, or to the first filename in the list if *i* is out of range.

:f	Display the current filename and line number.
q	Exit from *more*.

Bugs

Currently, *more* only works on files of 2048 lines of text or less. (The editors can only edit files with 300 lines of text or less).

See Also

grep(C), lpr(C), tail(C)

Author

Mark de Jong (mdj@pro-mdj.cts.com)

Name

mv - Move a file

Syntax

mv *file*... [*target*]

Description

Mv moves one or more files to another file or directory. The source file may be in another directory, in which case a path must be given. *Mv* is useful for renaming a file as well.

If the *target* is a directory, the source *files* are moved to files in the *directory* using their original filenames.

If the destination file does not exist, it is created, and the contents of the source files are moved.

If a source *file* is given, but no target *file*, the current working directory is assumed as the target area, preserving the original filename, though *mv* refuses to move a file to itself.

Example:

```
mv godel escher bach $/usr/mdavis
```

This moves three files from the current working directory to the directory $/usr/mdavis.

```
mv tycho brahe
```

In this example, ‘‘tycho’’ is simply renamed ‘‘brahe’’. If ‘‘brahe’’ existed before *mv* was used, it is deleted before the name change is done.

See Also

add(C), cp(C)

Name

news - System news bulletins

Syntax

news [*options*] [-t *item*] [*items*...]

Description

News displays and manages system news bulletins (not to be confused with USENET news). These bulletins are created by the system administrator reporting important news about the system.

Without arguments, *news* displays any items that are new since the user's last session. New items are considered new for the entire session. In other words, each time *news* is invoked, the same new items are shown until the user logs out.

An interactive mode (-i option) allows users and the administrator to browse all news items. The administrator can also create, delete, and edit items interactively.

News recognizes a number of command line options discussed below. News *item* arguments given in the command line can be the item's index number, its corresponding file name, or the subject that references the item. (When referencing an item by subject, enclose it within quotes if it contains spaces).

Options:

-a	All items. Causes *news* to consider all items new (unseen).
-c	Disable cancel keys. During the display of news items in non-interactive mode, cancel keys have no effect. This also forces a --More-- prompt at the end of each item causing the user to acknowledge it before continuing.
-i	Interactive mode. Various commands can be entered interactively. Type ? for a list of commands.
-n	News item list. This lists all items, showing their index numbers, file names, dates, and subjects.
-s	Reports only a count of the number of news items that exist.
-t item	Touches (updates) a news item, making it the newest item. This is useful for resurrecting an old item and making it new news.

While in interactive mode, only the super user can delete, edit, or add news items.

Notes

Editing a news item does *not* make it new.

News should be invoked whenever a user signs on. This is not done automatically by the *login* program, as in older versions of ProLine. See csh(C) for details on the *cshrc* script where an entry for *news -c* makes good sense.

Files

$/sys/news/* - news files.

See Also

csh(C), mcinews(C)

Name

od - Output file dump

Syntax

od [-a] *file*...

Description

Od displays the contents of a file in hexadecimal format, useful for inspecting the structure of binary files. The output produced appears as follows:

```
0000- 61946301 76DEADBE ...EFFEEDAF
0080- D21CA1FA 1FA20EDF ...D4C00BFC
```

By default, only hex data is shown in a tight format that displays 32 bytes-worth of data per line.

The -a option generates a looser format (16-bytes worth per line) along with an ASCII display next to each line.

Note

Od is traditionally ‘‘octal dump’’, but normal people avoid octal.

See Also

cat(C), inspect(C), tail(C)

Name

par - ProLine archive utility

Syntax

par [*startdir*] [*filelist*] [*parfile*]

Description

The *par* command allows you to archive up to 255 files into a single ProLine archive (.par) file. For example, typing:

```
     par $/etc/help $/etc/help update.par
```

packs all files in the $/etc/help directory to the file ‘‘update.par’’ in the current working directory.

The arguments are:

startdir	This is the path to the topmost directory containing files that will be archived. That includes files in subdirectories found below *startdir* in the directory hierarchy. Using $/ is usually a good starting directory.
filelist	This argument has special qualities. If it describes the path to a subdirectory, all of the non-directory entries in that subdirectory will be chosen for archiving. Note: this is not recursive. If *filelist* is a text file, it contains a list of filenames (full or partial paths) to indicate the files that should be archived. This file must contain one pathname entry per line. (The *find* command can generate such a file--and it can be used to create a list of files that descend through nested directories).
parfile	The *parfile* argument is the name of the archive file to create.

If any of these arguments are omitted from the command line, *par* prompts you for them, even the entry of the list of filenames.

File Lists

Note that the files you choose for archiving must follow certain rules. The most important is that you should never introduce a new path to a file that hasn’t had its parent directory entered beforehand. For example, it’s okay to select the file ‘‘booga’’ in ‘‘$/ooga/booga’’, so long as you chose ‘‘$/ooga’’ before it. Also, none of the entries can specify a file that resides outside of the *startdir* hierarchy.

Archives will not contain the full path to the original archived files. Therefore, the person doing the unarchiving must be sure about where the contents of the archive will be stored.

Note

Par currently creates archives in the Binary II format, but this may not always be the case. No compression is done.

See Also

find(C), unpar(C)

Name

passwd - Change login password

Syntax

passwd [*name*]

Description

Passwd changes (or installs) a password associated with the user *name* (your own name by default).

The program prompts for the old password and then for the new one, and the new password must be typed twice, to forestall mistakes.

New passwords must be at least four characters long if they use a sufficiently rich alphabet and at least six characters long if monocase.

Only the owner of the *name* or the root-user may change a password; the owner must prove he knows the old password.

If ‘‘none’’ is selected for the new password, the user will not be asked to enter a password when logging in (this is desirable for public accounts).

Files

$/etc/passwd - the password file.

See Also

adduser(ADM), eduser(C), gid(ADM)

Name

pc - Programmable calculator

Syntax

pc [*expr*]

Description

Pc is a programmable calculator. It provides simple or very sophisticated number crunching -- from basic adding-machine work to complex mathematical expression evaluation.

Features

- *Arithmetic Operators*. +, -, / (division), * (multiplication), ^ (exponentiation), nested parenthetical ordering.
- *Relational Operators*. =, >, <, <>, >=, <=, AND, OR, NOT.
- *Advanced Functions*. SGN, INT, ABS, SQR, RND, LOG, EXP, COS, SIN, TAN, ATN.
- *String Functions*. LEN, STR$, VAL, ASC, CHR$, LEFT$, RIGHT$, MID$.
- *User Defined Variables*. Up to 32 pre-evaluated or constant variables.
- *Stack*. Holds up to 32 values.
- *History*. A 100-entry command history allows you to select one or more previous entries for additional processing. The history also maintains the result of each operation at each point in time.
- *Precision*. Includes 38-digit floating point precision.

The functions adhere closely to the standard BASIC programming language syntax.

The Command Line

When invoked with the *expr* argument, *pc* evaluates the expression, displays the result, and exits. Example:

```
# pc 2 * 2 - 8
-4
# _
```

When invoked without an argument, *pc* enters a command mode, allowing you to issue multiple commands to do simple or complex calculations. This is the initial command line prompt:

```
pc [0] 1> _
```

The number in [square] brackets is the *current result register*. This register holds the value of the last operation that *pc* performed. The number before the '>' character is the line number of the next entry to be added to the history buffer.

Entering any number or numeric expression at this prompt causes *pc* to evaluate it and change the result register accordingly:

```
pc [0] 1> 6502 + 65816
pc [72318] 2> _
```

If the command begins with an arithmetic operator (+, -, /, etc.), *pc* performs the operation on the value contained in the result register. For example:

```
pc [72318] 3> - 65816
pc [6502] 4> _
```

Spaces are optional within expressions. They are needed only to delimit *pc* commands from their arguments, if any.

Boolean Math

Boolean arithmetic is accomplished by entering Boolean expressions on the command line:

```
pc [6502] 5> "foo" > "bar"
pc [1] 6> _
```

Any true expression sets the result register to one, while a false expression sets it to zero. Boolean operations against the result register are allowed by entering a relational operator as the first argument on the command line.

Precedence

Use (parentheses) to enclose operations to be performed first. Operations buried deepest within parenthesis (that is, parentheses inside parentheses) are performed first.

Pc's precedence rules are to collect and evaluate nested parenthetical expressions first, then perform multiplication and division (left to right), followed by addition and subtraction (left to right).

Commands

Pc recognizes the following commands:

quit	(or q). Quits *pc*.
clear	(or cl). Clears memory, including the stack and all variables. The result register is reset to 0.
history	(or hist). Displays a history of the previous commands entered and their results.
help	(or ?). Shows a list of the available commands.
do x[,y]	(or run). Tells *pc* to process history entries from x to y. If y is omitted, *pc* processes line x only. Intermediate results are shown after each entry is processed. All lines are processed, except for lines containing do (or run) commands.
set var=expr	(or let). Defines (or redefines) and sets the variable named var to the evaluated expression expr. The value of var is obtained by prefixing it with the character $. The result register is not affected.
define var=str	(or def). Defines (or redefines) the variable named var to the string str. The string contains an expression which is not evaluated until the variable is used later in an expression. The value of var is obtained by prefixing it with the character $. The result register is not affected.
print expr	Evaluates the expression expr and prints the result. The result register is not affected.
list	Lists the defined variables and their current values.
push [expr]	Pushes a value onto the stack. If no argument is given, the current value of the result register is pushed. If an expression argument is given, its value is pushed onto the stack. The result register is not affected.
pull [var]	(or pop). Pulls the last pushed value off the stack. If no argument is given, the value pulled off the stack is stored into the result register. If the name of a variable is given (whether previously defined or not), the value is stored in that variable, and the result register is not affected.
stack	Displays the contents of the stack. The next item off the stack is the last one listed.

Constants and Keywords

Pc comes with a constant for pi (3.141592653) referenced by the keyword PI. In addition, the result register can be accessed in expressions by using the keyword RESULT. Example:

```
pc [1] 6> result * 3 * 3 * result
pc [9] 7> _
```

Notice how the initial value of result is retained throughout the entire evaluation of the expression, resulting in 9. (1 * 3 * 3 * 1 = 9).

User Variables

You may define up to 32 variables for use in expressions. Variables are defined and initialized using the set (or let) and define (or def) commands.

Variables initialized with set have their expressions evaluated first, and the result is stored in the variable. That is to say, a variable defined with set will simply hold a number. Example:

```
pc [9] 8> set answer = asc("Life, the Universe, ...")-34
pc [9] 9> print $answer
     42
pc [9] 10> _
```

Variables initialized with define retain their expressions as a constant value. The result of the expression is evaluated at calculation time when the variable is given in an expression. Here are two simple examples:

```
pc [9] 10> define faren = 1.8 * result + 32
pc [9] 11> define celsius = (result - 32) / 1.8
pc [9] 12> list
     $answer    = 42
     $faren     = 1.8 * result + 32
     $celsius   = (result - 32) / 1.8
pc [9] 13> _
```

Unlike the variable answer, set in the previous example, faren and celsius retain their actual assignment rather than being evaluated. They also make use of the RESULT keyword so that their functions operate on the current value of the result register.

If the result register holds the value 9 degrees Celsius, then entering $faren on the command line will convert the value to its Fahrenheit equivalent:

```
pc [9] 13> $faren
pc [48.2] 14> _
```

Conversely, entering $celsius changes the value back to Fahrenheit:

```
pc [48.2] 14> $celsius
pc [9] 15> _
```

The Stack

The *pc* provides a 32-entry stack for temporary storage of numbers. You can push values onto the stack and pull them off in last-in, first-out (LIFO) order. Using the stack instead of declaring temporary variables can speed up multi-line computations.

Diagnostics

Typical diagnostic messages from *pc* are:

‘‘### syntax error’’
Instructions given are improperly formatted or not recognized.

‘‘### illegal quantity error’’
A value used with a function is out of range. For example, passing a value greater than 255 to CHR$() causes this error.

‘‘### overflow error’’
A computation involved a number out of the range of the program. *Pc* can accept numbers in the range of 1E-38 to 1E+38.

‘‘### division by zero error’’
Self-explanatory. This is not allowed.

‘‘### type mismatch error’’
A value passed to a function is not of the appropriate type. This occurs, for example, when passing a string to a numeric function, e.g., SGN(‘‘a’’).

‘‘### out of memory error’’
The variable table is full.

Author

Morgan Davis (mdavis@pro-sol.cts.com)

Name

plush - ProLine Users Shell

Syntax

plush

Description

Plush is a simple, flexible, and friendly shell environment for the typical ProLine user. It employs a menu-style interface offering access to resources that one would only be able to access from a more powerful shell, such as *csh*.

Using PLUSH

When *plush* is invoked, it presents a menu of commands. After a selection is made, the appropriate task is performed. This can involve launching an application or performing an internal function within *plush*, such as going into a submenu. Upon exiting a launched application, *plush* resumes, displaying the last menu presented.

Special keys: To back out of a submenu and return to a previous menu, press a ‘‘backup’’ key. A backup key is any control character other than the user’s cancel (interrupt) key. The backspace or RETURN keys are suggested as backup keys. Pressing a cancel key halts a process and returns to the current menu prompt. Pressing ‘?’ shows a list of a menu’s commands. The ‘+’ and ‘-’ keys toggle --More-- prompting.

Plush can ask for a line of input as an argument to some menu functions. For example, some disk utilities would require the name of a file to act upon. *Plush* recognizes the characters *$/* in inputs as a variable which expands to the ProLine system directory (just like in *csh*).

Plush Menus

Menus are defined in a *plush* menu file. When invoked, *plush* attempts to locate a menu file in this order:

1. A file named ‘‘plush.m’’ in the user’s $/adm directory.

2. A file named ‘‘plush.m.#’’, where # is the group ID of the user, in $tmpdir.

3. A file named ‘‘plush.m’’ in $tmpdir.

4. A file named ‘‘plush.m.#’’, where # is the group ID of the user, in $/etc.

5. A file named ‘‘plush.m’’ in $/etc.

This scheme allows the administrator to assign custom menu files to individual users, groups, and others.

Notes

Plush runs faster if it and its menu files are located on a RAM disk.

For details on designing custom menus, see plush(F).

Files

$/adm/*/environs - user's environment settings,
$tmpdir/plush.env - temporary environment file.

See Also

csh(C), plush(F), setenv(C)

Name

ps - Report process status

Syntax

ps [-defhmu] [-j *id*] [-k *id*] [-l *module*]

Description

Ps reports system module and active user processes. Without options, the -u flag is assumed.

Options are:

-d | -h	Select module information display format: decimal (-d) or hexadecimal (-h).
-e	Show everything. A list of user (-u) processes and system modules (-m) are both shown.
-f	Full information. Lists are displayed with as much information as possible. With this option, full pathnames are displayed with -u.
-m	Show modules. Each entry gives a module's index and ID number, size, address in memory, modification date, name and version number.
-u	Show user processes. (default)
-j id	Show information for one module where *id* is either its index or ID (in decimal).
-k id	Kill a module where *id* is either its index or ID (in decimal). See Warning. Super user only.
-l module	Load *module* by name. If the *module* name contains no slashes, it is assumed to reside in $/sys/modules. Super user only.

Warning

Killing system-dependent modules can have unfortunate results. Loading modules indiscriminately can cause serious memory errors.

Files

$/sys/modules/*

See Also

boot(ADM)

Name

pwd - Print working directory name

Syntax

pwd

Description

Pwd prints the pathname of the current working directory. This is identical to the following shell command

```
echo .
```

Upon logging in, the current working directory is set to what is called your ‘‘home’’ directory, the pathname contained in the shell’s $home variable. If you change directories with the *cd* command, use *pwd* to report the new working directory.

See Also

cd(C), csh(C)

Name

resume - Resume file generator

Syntax

resume

Description

Resume automatically produces a resume file for a user by asking a few questions that have been defined by the System Administrator in a resource file. The user answers the questions, and a resume file with that data is generated and stored in the user's home directory.

System Administrator Notes

A resource file, $/etc/rsrc/resume.rsrc, contains information for the *resume* program. This resource file can be created and changed using one of the online ProLine editors. A sample resource file follows:

```
5
Please enter some information about yourself.
Age:
Male or Female:
Occupation:
Computers Owned:
Hobbies/Interests:
```

The first line gives the number of questions in the file. The second line is a message directed to the user. The balance of the file contains questions to be answered. Each question begins on a new line.

Files

$/etc/rsrc/resume.rsrc - resume data file,
$/usr/*/resume - user resume files.

Author

Bob Lindabury (bobl@bobsbox.rent.com)

Name

rm, rmdir - Remove files or directories

Syntax

rm *file*...
rmdir [-p] [-s] *dir*...

Description

Rm removes the entries for one or more files from a directory. Removal of a file requires ‘‘destroy’’ permission in its directory, but neither read nor write permission on the file itself.

If a designated file is a directory, it can only be deleted by *rm* if it is empty.

Rmdir removes entries for the named directories, including all files and subdirectories within the directory itself. If the *-p* option is given, the name of each file removed is printed first. If the *-s* option is given, the named directory is purged, but the directory entry itself is retained.

See Also

mv(C)

Name

rot13 - Rot13 file conversion utility

Syntax

rot13 [-h] *infile* [*outfile*]

Description

Rot13 encodes (or decodes) a text file, rotating each alphabetical character in the file 13 places in the alphabet. Encoding a text file with *rot13* prevents reading the text without the special effort required to decode it. For example, offensive jokes in the rec.humor.funny newsgroup are encoded using *rot13*.

Since the *rot13* encoding is also self-decoding, a file encoded into *rot13* format is decoded in the same manner. A second pass over the file decodes it.

The optional -h flag skips the header on the *infile*. This prevents *rot13* from processing any initial header lines, up to the first blank line, processing only the body of a message. In other words, header lines are left ‘‘as-is’’.

Rot13 prints its output to the screen, or to the file given as the *outfile* argument.

Examples:

```
rot13 string.joke
```

processes the file ‘‘string.joke’’ with output appearing on the screen.

```
rot13 -h string.joke joke.rot
```

processes ‘‘string.joke’’ into the file ‘‘joke.rot’’, leaving the header alone.

See Also

safecom(C), uudecode(C), uuencode(C)

Author

Dean Fick (dean@pro-electric.cts.com)

Name

rx - Receive files using XMODEM or YMODEM

Syntax

rx [*options*] [*file*...]

Description

Rx receives files using XMODEM or YMODEM transfer protocol.

If *rx* is used for receiving files with XMODEM, it invokes an XMODEM receive for each *file* given on the command line. Files are assigned the names given in the command line in the order in which they are received.

If YMODEM batch is used to receive files, the *file* argument(s) are optional. If given, however, the files received are assigned the names given on the command line, rather than those supplied by the sender.

A number of options control the transfer characteristics:

-b Binary II mode (Apple II). This causes *rx* to unwrap files in Binary II format. This restores directory information unique to the file, and required by the ProDOS operating system.

-c 128-byte blocks with CRC-16.

-d Double delay tolerance. This effectively doubles the time that *rx* allows for responses over sluggish packet-switched networks.

-f Accept full pathnames from the sender -- the prefix to the file plus the file's name.

-k 1K blocks with CRC-16.

-l 4K blocks with CRC-16.

-p ProDOS mode (Apple II). Like the Binary II mode, this option employs an extension unique to some Apple II terminal programs to transfer file information. This scheme has been made obsolete by the Binary II standard, but is included for compatibility with older programs.

-r Replace existing files.

-t Text mode. This mode converts end of line characters (newlines) to be compatible with all operating systems. Include this option whenever text files are received to ensure proper newline conversion. Newline characters are Control-M in Apple II and Macintosh operating systems, Control-J in UNIX, and Control-M Control-J in MS-DOS.

-v Verbose 1K and 4K modes. When the -k or -l modes are used, -v causes them to initiate the transfer by sending ‘‘CK’’ or ‘‘CKL’’ to the sender. Normally, only ‘C’ is sent.

-y YMODEM batch mode receive with at least 1K blocks and CRC-16. YMODEM-g mode is not supported for receiving.

Without options, *rx* receives files in plain 128-byte XMODEM protocol using checksums for error correction.

See Also

rz(C), sx(C), sz(C)

Name

rz - Receive files using ZMODEM

Syntax

rz [*options*] [*file*...]

Description

Rz receives files using the ZMODEM transfer protocol. The *file* argument(s) are optional. If given, however, the files received are assigned the names given on the command line, rather than those supplied by the sender.

A number of options control the transfer characteristics:

-d	Double delay tolerance. This effectively doubles the time that *rz* allows for responses over sluggish packet-switched networks.
-f	Accept full pathnames from the sender -- the prefix to the file plus the file's name.
-r	Replace existing files.
-t	Text mode. This mode converts end of line characters (newlines) to be compatible with all operating systems. Include this option whenever text files are received to ensure proper newline conversion. Newline characters are Control-M in Apple II and Macintosh operating systems, Control-J in UNIX, and Control-M Control-J in MS-DOS.

See Also

rx(C), sx(C), sz(C)

Name

safecom - File encryption utility

Syntax

safecom *mode infile code* [*outfile*]

Description

Safecom, designed to help maintain safe communications channels, allows you to encode and decode text files. Encoding and decoding is performed by means of a *code* key (password) of your own choice. You must remember the *code* to be able to work with encoded files.

The *mode*, either -e for encoding or -d for decoding, tells *safecom* how to process the *infile* based on the *code* given. All output is sent to your terminal unless you specify the optional *outfile*.

Note

This program is an adaptation of the original SAFECOM by Bill Parker (‘‘Hard Talk’’ columnist for ‘‘Softalk Magazine’’, R.I.P.). It was rewritten to operate under the ProLine shell environment by Morgan Davis.

This program is slow. It can take quite a while to process a large file.

See Also

rot13(C), uuencode(C), uudecode(C)

Name

sendmail - Send mail over the internet

Syntax

sendmail

Description

Sendmail sends a message to one or more people, routing the message over whichever networks are necessary to deliver the message to the correct place. Other programs provide user-friendly front ends; *sendmail* is used only to deliver preformatted message files placed in the $/spool/mail subdirectory.

Routing Strategy

Local user addresses are determined by matching user directories in $/usr. Remote site addresses are determined by matching directories in $/mdss.

If a local user address is unrecognized, an alias file ($/etc/aliases) is searched and the route is aliased appropriately if a match is found. If a remote site address is unrecognized, it is searched for in a path alias database ($/etc/paths) and the bogus route is substituted if an alias is found. If a route still fails, its appended domain, if included, is searched for in the path alias file. If a domain match is found, the path is aliased appropriately, otherwise the letter is forwarded to the authority site (governed by the system's domain).

If a local address in an alias list begins with '~' (tilde), it forces sendmail to write into the named mailbox without verifying that the address is an actual user on the system. See *aliases(F)* for details.

If *sendmail* is unable to find a valid route at this point, the message is forwarded to a ''smart host'' (see below). If no smart host is designated, the message is returned to the sender with an error message.

Special Handling

Sendmail also resolves local letters for users who have a *forward* file in their home directory. The *forward* file contains e-mail addresses that replace the local user's address, thus forwarding letters to addresses other than (or including) the local user. See forward(F) for more details.

It is also the responsibility of *sendmail* to deliver local messages with attached binary segments, as formatted by the *rcp* program. Text portions of such messages are delivered in the usual fashion while the binary portion is written into the addressee's user area. If a file collision occurs, the file is saved with a unique name.

Resource File

The resource file for sendmail contains entries that regulate its operation. Each entry is on a line by itself in the following order:

Line 1	Smart host. This line contains the complete path to a ‘‘smart host’’, that is, one that can handle unresolved addressing. For example, this line might contain ‘‘pro-neighbor’’. Any messages that can’t be resolved locally are eventually passed on to ‘‘pro-neighbor’’, a smart host that attempts to deliver the message.
Line 2	Busy flag. This line contains a number (0 for FALSE, and 1 to TRUE) to determine if the modem should be placed off hook (busy) while *sendmail* delivers mail.
Line 3	Progress level. This line contains a number (0 to 9) setting the level of progress reporting while *sendmail* runs. Level 0 is minimal reporting, while 9 is maximum reporting. *Sendmail* displays a code for each letter delivered. These codes are: (.) = local letter (!) = offsite letter (>) = local *rcp* letter (F) = forwarded letter (N) = news batch (X) = bounced letter
Line 4	Aliases location. This line holds the pathname to the *aliases* database file. See *aliases(F)* for details. Default is $/etc/aliases.
Line 5	Paths location. This line holds the pathname to the *paths* database file. See *map(F)* for details. Default is $/etc/paths. By setting the pathnames of the aliases and paths files to a RAM disk location, sendmail’s performance can be increased. Be sure to place copy commands in $/etc/rc to copy the files to the RAM disk after starting up. See *rc(ADM)* for details.
Line 6	Favor smart host flag. This line contains a number (0 for FALSE, and 1 to TRUE) to determine if *sendmail* should attempt to locate addresses containing aliases in the paths database. If set to 1, *sendmail* consults the paths database to lookup addresses with domains that are not the same as the host’s. This is useful when the host communicates with more than one domain authority. If set to 0, any addresses with domain specifiers other than the host’s are passed immediately to the smart host. This avoids extra passes through the paths database which speeds up processing.

If a fatal operating system error occurs, *sendmail* displays error information, followed by an alert sound at the console. This will last for approximately a minute before the system is restarted. Press Control-C to cancel the error and enter BASIC.

Files

$/etc/aliases - user alias database,
$/etc/paths - path alias database,
$/etc/rsrc/sendmail.rsrc - resource file,
$spool/mail - mail spool directory,
$/usr/*/forward - user’s forwarding address file.

See Also

aliases(F), forward(F), mail(C), mdss(NET), rcp(NET)

Name

server - Internet file server

Syntax

server

Description

Server processes any file server requests found in the server's mailbox file. Requests are made by sending mail to server@host, putting the request in the Subject field of the message. If the ‘‘Subject:’’ field is blank or not found, the request is assumed to be in the first line of the body of the message.

Request commands are:

DIR [dir ...]	Request directory listings
HELP	Request help on using the server
INDEX [dir ...]	Request index files for the server directories
MAP [host ...]	Request a site map from the host's archives
SEND [dir/]file ...	Request a file from the server's archives

The optional [dir] argument (without the brackets, of course) is the name of a subdirectory area within the server's main directory. The user needn't specify a complete path to the directory, only the directory's name. Example:

```
index apple2
```

Should the directory itself contain another directory, the user can access it by using a slash to separate the two (e.g. ‘‘index apple2/programs’’).

The server commands (except for HELP) can take multiple arguments. Example:

```
send file1 file2 dir/file3
```

If the MAP command is given without an argument, a directory listing of the map archive area is sent.

Server Default Directories

By default, the server's domain includes $/pub and all the directories within it, and the site maps are assumed to be found in $/sys/maps. The administrator may choose different paths to these areas by editing the server resource file (*$/etc/rsrc/server.rsrc*) and placing the complete path to the main server directory in the first line of the file, and the complete path to the map directory in the second line.

Notes On Using INDEX and DIR

When these commands are used without any arguments, the entire structure within the main server directory is referenced. This means that if DIR alone were specified, the user would get directories for all the subdirectories inside of the server's domain. Likewise, INDEX alone would get indexes for all server areas.

When a directory argument is used with DIR or INDEX, the structure including that directory and its domain is referenced. In short, these commands return the entire tree (if no argument is given), or branches of the tree depending on the argument given. Most server archives have directory structures only a level or two deep.

Setting Up The Server

As stated, the server's domain is the $/pub directory and all files and subdirectories within it. If your server archives contain numerous files it is best to organize them into subdirectory areas.

Each area (including $/pub) should contain a file named INDEX which gives a quick table of contents for that area. The INDEX file format is not rigid, though it should at least contain this information:

- o Name of the host system
- o Name of the area
- o Date when the index was last modified
- o Contents of the area

Typically, the content list contains entries, one per file, giving the name of the file and a short description. You can add more information if you desire, though it is best to be as brief as possible, giving only the information necessary for browsing.

Access Protection

The server abides by the permission attributes assigned to directories in the server archives. If a user requests a file from a directory that does not have read permission, the request is stored in the queue file called $/etc/server.req (which can be used as a shell script later on). The root user can validate each entry, removing those without clearance, then launch the file as a shell script in order for the file requests to be fulfilled. Example:

```
source server.req
```

The file's type is text, and requires the shell's ‘‘source’’ command in order to run.

Server Message

Every outgoing file sent by the server can also include an optional message. This message is stored in the $/etc/help/server.msg file. This could include short messages such as rules regarding server usage, and so on. It should probably end with a dashed ‘‘cut here’’ line to let the requester know when the real file starts.

Server Log

The server maintains a log file ($spool/logs/servlog) listing the requests and when they were processed.

Files

$/pub - server’s domain,
$/sys/maps - site map archive area,
$/sys/mail/server - server’s mailbox,
$spool/logs/servlog - server log file,
$/etc/server.req - server requests pending validation,
$/etc/help/server - server help file,
$/etc/help/server.msg - server message file,
$/etc/rsrc/server.rsrc - server’s resources.

See Also

rcp(NET), sendmail(C)

Name

set, unset - Set or unset shell variables

Syntax

set [*name*=*value*...]
unset *name*...

Description

With no arguments, *set* lists all shell variables and their values. With arguments, *set* assigns a *value* to the variable *name* given.

Example:

```
set hello=world
```

Sets the variable *hello* to the value ‘‘world’’. To access the value in this new variable, prefix it with a dollar sign, as in:

```
1% echo $hello
world
2% ls $hello
world: directory not found
```

Multiple variables can be assigned all at once, as with:

```
set foo=bar seven=7 hard="disk drive"
```

To erase a variable, the *unset* command is used with one or more variable names:

```
3% unset hello
4% unset crash bang bigbang bam
```

Note that variables used in the *value* portion of a *set* command are expanded into the *value*. This may not be desirable. In the case where the variable should not be expanded during the assignment, escape the variable with a backslash (\):

```
set prompt="[\$cwd] "
```

See Also

csh(C)

Name

setenv - Set environment

Syntax

setenv [-u user]

Description

Use *setenv* to bring up an interactive menu of options allowing you to define various environmental preferences for your sessions online. *Setenv* updates your current settings, and will also store them for future sessions. Each time you log onto the system your environment is initialized as defined.

Options that you can modify:

- o Cancel key
- o Null delays after newlines
- o Tabs: preserved or expanded
- o Terminal height in lines
- o Terminal width in columns
- o Terminal to emulate
- o Menu mode, or just prompts
- o Single-key input mode: on or off
- o Choice of text editor (standard or visual)

The -u option allows the super user to specify a different user.

Files

$/adm/*/environs - user's environment settings.

See Also

stty(C), tset(C)

Name

setfile - Set file attributes

Syntax

setfile [*options*] *file*...

Description

Sets attributes for one or more *files*. The *options* apply to all files listed.
Options are:

-a flags
: Sets or clears the file attributes. The *flags* string is composed of the characters listed below. Attributes that aren't listed remain unchanged.

D = Destroy enable
N = Rename enable
B = Backup needed
I = Invisible
W = Write enable
R = Read enable

Uppercase letters set the attribute to 1; lowercase letters clear it to 0.
For example,

```
setfile -a dnwbR Filename
```

Clears the Destroy, Rename, Write, and Backup bits, and sets the Read bit. This means that the file cannot be deleted, renamed, or written to, but it can be read from. The backup bit is also cleared, which tells backup programs that the file doesn't need to be backed up.

-at type
: Sets the auxilliary type. *Type* may be specified either in decimal or hexadecimal form. Hexadecimal numbers must be preceded by a '$' character or ''0x'' character sequence.

-c date
: Set the creation date. *date* is a string in one of the forms listed:

```
mm/dd/yy
"mm/dd/yy hh:mm"
"mm/dd/yy hh:mm AM"
"mm/dd/yy hh:mm PM"
```

representing the month (1-12), day (1-31), year (0-99), hour (0-23), and minute (0-59). The string must be quoted if it contains a space. A period (.) indicates the current date and time.

-m date
: Set the modification date. *Date* is the same format as for the -c option. A period (.) indicates the current date and time.

-t type
: Sets the file type. *Type* may be specified as a decimal number, a hexadecimal number when preceded by a '$' or ''0x'' character(s), or a 3-character filetype mnemonic.

-s size	Sets the file's size in bytes. *Size* may be specified as a decimal number or a hexadecimal number when preceded by a '$' or ''0x'' character(s).

Examples

```
setfile -a b -m \. foo.asm
```

Clears the backup-needed access attribute bit, and set's the modification date to the current date and time on the file ''foo.asm''. Note that the shell interprets a period as a reference to the current working directory, thus the backslash (\) is used to include the period for *setfile*.

```
setfile -m "07/03/90 02:25" mbox
```

Sets the modification date to July 3, 1990, and 02:25 A.M. on ''mbox''. Note that the date string is quoted, since it contains a space, and that AM is assumed when the hour is less than 13 and no AM or PM is specified.

```
setfile Foo -t bin Bar -s 4096 Baz -at 0x4000
```

Operates on the files Foo, Bar, and Baz, setting their types to BIN ($06), auxiliary types to $4000, and their sizes to 4096 ($1000 or 0x1000) bytes. Note how filenames and options can be intermixed.

See Also

dstat(C), ls(C), sweep(C)

Name
sleep - Suspend execution for an interval

Syntax
sleep *time*

Description
Sleep suspends execution for *time* seconds. It is used to execute a command after a certain amount of time as in:

```
sleep 105; command
```

Note
Non-root processes cannot suspend execution for more than 600 seconds (ten minutes).

Name

sm - Send modem command

Syntax

sm [*options*] [*command*]

Description

Sm sends a modem *command* to the external modem attached to the computer. This is handy for putting the modem into certain states, or temporarily adjusting the modem's characteristics beyond those installed during system initialization.

Sm recognizes the following *options*:

-h	Hangup. Tells the modem to disconnect.
-p	Pickup. The modem is instructed to lift the phone off-hook, making it appear to be busy.
-r	Reset. This initializes the modem.
-s mode	Special. This activates system-specific features of the modem based on a *mode* value. Current modes are: 3 = Turn off the modem's speaker completely 4 = Speaker is on until a connection is made 5 = Speaker is on all the time 6 = Resets the modem (same as the -r option) 7 = Force serial port to always report carrier 8 = Serial port reports the actual carrier signal 9 = DTR line signal is lowered 10 = DTR line signal is raised

See Also

startup(ADM)

Name

sort - Sort a file

Syntax

sort *file*...

Description

The *sort* command allows you to view the alphabetized contents of a file. Example:

```
        sort vivaldi
```

This displays the contents of the file ‘‘vivaldi’’ in alphabetical order. If the last argument starts with ‘>’, the sorted lines are written into that file.

Note

Sort handles files with 400 lines of text or less.

See Also

grep(C), split(C), wc(C)

Name

source - Read shell commands from a file

Syntax

source *file*

Description

Source causes the shell to read lines from a *file* and execute them one at a time as a batch operation. Processing terminates when the end of the file is reached, or when appropriate commands instruct the shell to stop.

Source reads commands from any type of file; therefore, there is no need to change the file's type.

To invoke a shell script without having to use the *source* command, change the file's type to ‘‘CMD’’ with *setfile*.

See Also

csh(C), scripts(M), setfile(C)

Name

split - Split a file into pieces

Syntax

split [-*lines*] *file* [*name*]

Description

Split reads *file* and writes it into pieces, as many as necessary, onto a set of output files. The size of each piece is given in lines by the *lines* argument (or 1000 lines by default). The name of the first output file is *name* with *aa* appended, and so on lexicographically. If no output name is given, *x* is used.

Note

Split refuses to overwrite any existing files.

See Also

add(C), wc(C)

Name

stty - Set terminal options

Syntax

stty [*option*...]

Description

Stty is used to set certain I/O options on the current output terminal. With no argument, it reports the current terminal settings. The following options are recognized.

intr ^c	sets your interrupt cancel key (default: Control-C). It must be a control character or DEL. The argument to *intr* begins with a caret (^) followed by the letter corresponding to the control character (e.g., ^C for Control-C). To specify DEL, use ^?.
nulls 0	sets the number of nulls after a newline sequence (default 0). The argument to *nulls* is an integer from 0 to 255, which specifies the number of null characters to send.
lines 23	sets the number of lines your terminal screen has (minus one) and turns on screen paging. When the specified number of lines have been displayed, you'll then be prompted with --More-- at the bottom of the screen. Pressing any key will resume the display. You can turn paging off by using "stty lines 0".
-tabs	tab characters are converted to spaces before printing. (default)
tabs	tab characters are preserved.

Stty can be used with any or all of the options described above on the same line. For example:

```
stty intr ^? nulls 3 lines 22 -tabs
```

This tells the system to recognize the DEL key as the interrupt character, send 3 nulls after each line, turn screen paging on for every 22 lines of text displayed, and expand tabs to spaces.

See Also

setenv(C), tset(C)

Name

sweep - File utility

Syntax

sweep [*directory*]

Description

Sweep is patterned after the popular CP/M SWEEP utility which allows you to manipulate files with much ease. *Sweep* lets you to perform many different disk-related operations on single files as well as on batches of files.

To operate on a single file, simply advance to the file entry you desire and then press a command key.

To operate on a batch of files, advance to each file desired, "tag" it, and then press the letter of the command you want to perform on those files.

When *sweep* is invoked it first displays a help screen, and then reads all the filenames from the current working directory. If you use a directory name with *sweep*, as in *sweep $/pub/stuff*, the working directory is changed to the new directory.

At this point you may use any of the following commands:

T	tag a file (use control-T to tag all files)
U	untag a file (use control-U to untag all files)
D	delete the selected file(s)
C	copy the selected file(s)
M	move the selected file(s). This is the same as a copy followed by a delete.
V	view the selected file(s).
P	go to previous file in the list.
^P	go to the previous directory. After pressing ^P you can type RETURN to select the prompted directory, or edit the path presented to you to go to a different area.
L	change to a new directory location. If you're currently sitting next to a subdirectory entry in the file list, pressing L RETURN changes to that directory.
A	alters protection attributes for the selected file(s). This allows you to ‘‘lock’’ or ‘‘unlock’’ a file.

S	toggles *sweep*'s list sorting feature. *Sweep* comes up initially with all the files in the list sorted alphabetically.
E	edit the selected file(s). The editor you select using the *setenv* command is used.
R	rename the selected file(s).
Q	quit *sweep* and return to the shell.
N	create a new directory in the current directory. Enter the name of the new directory when prompted.
?	display the command list

Any key other than those listed will advance you from the current entry to the next one. If you're at the last entry in this list, advancing will take you back to the first entry. Similarly, retreating beyond the first entry will take you to the last entry in the list.

When supplying the target path for a copy or a move, simply give the name of a directory if more than one file is tagged. If no files are tagged, you can either give the name of a directory, or supply a new path and filename to copy it with a different destination name.

See Also

cat(C), cp(C), ls(C), mv(C), rm(C), setfile(C)

Name

sx - Send files using XMODEM or YMODEM

Syntax

sx [*options*] *file...*

Description

Sx sends files using XMODEM or YMODEM transfer protocol. A number of options control the transfer characteristics:

-b	Binary II mode (Apple II). This causes *sx* to wrap files in Binary II format. This transmits directory information unique to the file, and required by the ProDOS operating system.
-c	128-byte blocks with CRC-16.
-d	Double delay tolerance. This effectively doubles the time that *sx* allows for responses over sluggish packet-switched networks.
-f	Send the file's full pathname -- the prefix to the file plus the file's name.
-k	1K blocks with CRC-16.
-l	4K blocks with CRC-16.
-p	ProDOS mode (Apple II). Like the Binary II mode, this option employs an extension unique to some Apple II terminal programs to transfer file information. This scheme has been made obsolete by the Binary II standard, but is included for compatibility with older programs.
-t	Text mode. This mode converts end of line characters (newlines) to be compatible with all operating systems. Include this option whenever text files are sent to ensure proper newline conversion. Newline characters are Control-M in Apple II and Macintosh operating systems, Control-J in UNIX, and Control-M Control-J in MS-DOS.
-y	YMODEM batch mode with at least 1K blocks and CRC-16. YMODEM-g is supported if requested by the receiver.

Without options, *sx* sends files in plain 128-byte XMODEM protocol using checksums for error correction.

See Also

rx(C), rz(C), sz(C)

Name

sz - Send files using ZMODEM

Syntax

sz [*options*] *file*...

Description

sz sends files using the ZMODEM transfer protocol. A number of options control the transfer characteristics:

-d Double delay tolerance. This effectively doubles the time that *sz* allows for responses over sluggish packet-switched networks.

-f Send the file’s full pathname -- the prefix to the file’s location plus the file’s name.

-t Text mode. This mode converts end of line characters (newlines) to be compatible with all operating systems. Include this option whenever text files are sent to ensure proper newline conversion. Newline characters are Control-M in Apple II and Macintosh operating systems, Control-J in UNIX, and Control-M Control-J in MS-DOS.

See Also

rx(C), rz(C), sx(C)

Name

tail - Deliver the last part of a file

Syntax

tail [+|-*number*[*lbc*]] *file*

Description

Tail copies the named file to the standard output beginning at a designated place.

Copying begins at distance +*number* from the beginning, or -*number* from the end of the file. *Number* is counted in units of lines, blocks or characters, according to the appended option *l, b* or *c*. If no *number* is specified, -10 is assumed. When no *unit* is specified, counting is by lines.

Examples

```
tail logfile
```

Displays the last 10 lines in logfile.

```
tail +10 logfile
```

Displays lines from 10 through the end of logfile.

```
tail -256c logfile >lastpage
```

Writes the last 256 characters in logfile to the file lastpage.

See Also

cat(C), grep(C)

Notes

Various kinds of anomalous terminal behavior may happen with binary data files.

Name

trim - Trim mailbox headers

Syntax

trim

Description

Trim reads through your mailbox file and copies its contents into a file in your home directory called ‘‘mail.t’’ (if it already exists, *trim* appends new letters to it). While copying your mailbox, *trim* removes all header fields except for the following which start with:

```
From
Date:
From:
Subject:
To:
Cc:
Bcc:
```

This significantly reduces the size of Internet-originated messages which contain a number of header lines. When *trim* has finished pruning your mailbox, it asks if you want it deleted. Normally, you would want it removed, as your mail has been copied into ‘‘mail.t’’ in your directory.

To work with ‘‘mail.t’’ file as your mailbox, invoke the mailer as follows:

```
mail -m mail.t
```

See Also

mail(C)

Name

tset - Set terminal emulation

Syntax

tset *termcap*

Description

Tset installs a terminal capability file (*termcap*) into memory for use with programs that require special emulation functions, such as being able to move the cursor, clear the display, turn on inverse video, and so forth.

The argument is the name of a termcap file residing in the directory $/sys/termcaps. As shipped, this system has many popular terminals to choose from, though your system administrator can create customized terminals for you if needed.

See Also

setenv(C), stty(C)

Name

unpar - ProLine archive unpacker

Syntax

unpar [options] *file* ...

Description

Unpar unpacks the contents of a ProLine archive (*par*) file. For example:

```
unpar update.par
```

unpacks the contents of the file ‘‘update.par’’ into the current working directory. Multiple par files can be unpacked by specifying them on the command line.

Options

-d	Causes any directories created to be set to the current working directory’s permissions. This is always the case for non-super users. Without the -d option, directories created under super user access are set to the permissions stored in the archive’s entry.
-l	Lists the contents of the archive. Nothing is extracted.
-r	Replaces existing files without warning.

If a file being unpacked already exists and the *-r* flag is not given, you are prompted to replace the existing file. Press Y to replace it, or N to skip it. Pressing R replaces the existing file and turns on the *-r* flag for you, causing subsequent collisions to be replaced automatically. Press Q to stop immediately and quit.

Note

Unpar can unpack any file in the Binary II format.

See Also

par(C)

Name

uuencode, uudecode - Encode/decode a binary file for mailing

Syntax

uuencode *source target*

uudecode *file*

DESCRIPTION

Uuencode and *uudecode* are used to send binary files via uucp (or other) mail. This combination can be used over indirect mail links.

Uuencode takes the named source file and produces an encoded version to the target file. The encoding uses only printing ASCII characters, and includes a mode of 644 and the target name for recreation on the remote system.

Uudecode reads an encoded file, strips off any leading and trailing lines added by mailers, and recreates the original file with the specified mode and name.

The encode file has an ordinary text form and can be edited by any text editor to change the mode or remote name.

Notes

The file is expanded by 35% (3 bytes become 4 plus control information) causing it to take longer to transmit.

Not all file information is retained, therefore it is suggested that files first be archived with compression to make up for the uuencoding expansion before being encoded.

The user on the remote system who is invoking uudecode must have write permission on the specified file.

See Also

par(C), rcp(NET), unpar(C)

Name

vedit - Visual text editor

Syntax

vedit *file*

Description

Vedit invokes a full screen editor on a text (txt) or shell script (cmd) file. Adequate terminal emulation is required to use *vedit*.

Control-Key Editing Commands

^M	Carriage return
^H	Backspace
^U	Cursor right
^K	Cursor up
^J	Cursor down
DEL	Delete left
ESC	Bring up the command line
^L	Redraw screen
^A	Start of line
^E	End of line
^D	Delete forward
^O	Toggle overstrike/insert mode
^Y	Yank line (cursor to right) to kill buffer
^X	Delete entire line into kill buffer
^P	Previous screen
^N	Next screen
^W	Toggle word-wrap
^T	Top of buffer
^B	Bottom of buffer
^G	Get kill line, insert at cursor's position
^I	Insert a tab
^F	Toggle tab mode (insert real tabs or spaces)

The Command Line

Pressing ESC will bring up the command line which recognizes the follow commands (invoked by pressing the first letter of a command's name):

New	Clear the buffer (with the option of saving any changes)
Save	Saves the buffer to the file.
Write	Writes the buffer to a different file.
Append	Appends a file to the end of the current buffer.
Find	Finds a pattern of text.
Replace	Replaces all occurrences of a pattern.
Info	Information on the current buffer's contents.
Quit	Quit the editor (with the option of saving any changes).

See Also

ed(C), edit(C), setenv(C)

Name

version - Report version information on programs

Syntax

version [-hi] [*path*...]

Description

Version generates a report of program version information. The report includes the official name of the program, version number, modification date, and author’s name.

If *path* references a file, *version* generates a report for that file. If *path* references a directory, a report is produced for all the files in that directory level. A typically useful invocation is

```
version $path
```

which generates a report for all programs found in the executable search path.

Version information is obtained in the first 255 bytes between ‘‘@(#)’’ markers. Therefore, entries are reported only for files that contain a version identification string within the first 255 bytes of the program or shell script.

Normally, *version* ignores files that do not include an indentification mark. Including the -*i* options instructs *version* to indicate files lacking version information.

The -*h* option disables printing of the report header.

Report output can be redirected into files by including >*file* at the end of the command line.

Without arguments, *version* displays the ProLine software version information.

See Also

whatis(CT)

Name

wc - Word count

Syntax

wc [-lwc] *file*...

Description

wc counts lines, words and characters in one or more *files*. A word is a maximal string of characters delimited by spaces, tabs, or newlines.

The -l option gives a line count. The -w option gives a word count. The -c flag gives a character count. If no option is given, the default is to count the lines, words and characters (-lwc).

See Also

grep(C), tail(C)

Name

whereis - Locate source, binary, and/or manual files

Syntax

whereis [-sbm] *name*...

Description

Whereis locates source files, programs, and sections of the manuals for specified files. The supplied *names* are first stripped of leading pathname components and any trailing extension of the form ‘‘.ext’’; e.g., *.c*. *Whereis* then attempts to locate the desired program in a list of standard places. If any of the *-b*, *-s*, or *-m* option flags are given, *whereis* searches only for the designated option or combination of options; binaries, sources or manual sections, respectively.

Example

The following example finds all files associated with the Conference System:

```
whereis -sbm cs
```

Files

$/etc/rsrc/man.rsrc - order and names of manual sections

See Also

find(C), man(CT), whatis(CT)

Name

who - Who is on the system

Syntax

who [*pattern*] [>*outputfile*]
who am i

Description

Who with no arguments displays a full listing of all the members who have accounts on the system. Shown are their login names, real names, locations, and last login times.

If *who* is given a *pattern* as a search condition, it prints out only those entries in the member list where the *pattern* is matched. This is normally used for selection of a single user by using their login name, but you may also find other parts of the entry such as cities, states, or last login dates and times. If a *pattern* requires space characters, enclose it in quotes.

When *who* finds an entry whose login matches the *pattern*, it displays that user's *resume* file if it exists in their directory. If the *pattern* given is ‘‘all’’, *who* displays a full member list and attempts to print out the resume files for every member.

If the last argument given is a file name beginning with a ‘>’, output from *who* is directed into the file.

‘‘who am i’’ displays your Internet address and full name.

Files

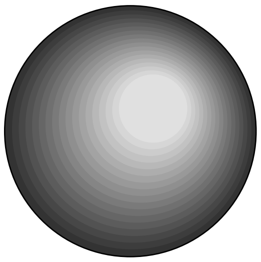
$/etc/passwd - password file,
$/etc/adm - account database,
$/usr/*/resume - a user's resume file.

See Also

eduser(ADM), gid(ADM), log(C)

ProLine

Typesetting Commands



Name

makebook - Create a file containing the entire manual

Syntax

makebook *bookfile* [-d] [*options*]

Description

Makebook, a shell script, builds a file containing the entire online manual for subsequent downloading and/or printing.

By default, *bookfile* is formatted for a PostScript printer (*manps* is used). Use the -d option to create a dot-matrix formatted book (*man* is used).

Makebook automatically requests progress (-p) reporting and assigns titles (-t) when generating output. Include *options* to pass additional options to *man* or *manps*.

Note

Since the file created by *makebook* will be uncommonly large, this script is not accessible to non-root users by default. To curtail users from creating their own books, it is suggested that the administrator create a single *bookfile* and place it into a public area (e.g. $/pub) so users may download it and print it offline to their own printers.

See Also

man(CT), manps(CT)

Name

man - Prints pages in the manual

Syntax

man [*options*] [*section*] *title* ...

Description

Man locates and prints pages for the named *title* from the designated *section* in the ProLine Reference Manual.

Since commands are given in lowercase, the *title* is always entered in lowercase. *Section* names are always uppercase. If no *section* is specified, the entire manual is searched for a matching *title*, and the first occurrence found is printed. You can search through a group of sections by separating the section names with colons (:) on the command line.

Man can display all entries for a specified section by using an asterisk (*) as the *title*. A *section* argument must be given.

Options

The following *options* are recognized by *man*:

-a	‘‘All’’ mode. Displays matching titles found in all sections of search list.
-d	Formats output for a dot-matrix printer.
-l rows	Sets the length of your output device in rows.
-p	‘‘Progress’’ mode. Prints the pathname of each title being processed.
-t title	Assigns a title to the header (e.g. -t "Local Commands"). This is mostly useful when printing an entire section.
-w cols	Sets the width of your output device in columns.

Section Names

The names and general descriptions of the available manual sections are:

ADM	Administrator’s Utilities
C	Commands
CT	Text Processing Commands

CP	Programming Commands
F	File Formats
G	Games
HW	Hardware Dependent
LOCAL	Local Utilities
M	Miscellaneous
NET	Network Commands
S	Subroutines and Libraries

You can add other section names as you desire. Each new section, however, must follow the standard section directory structure.

$/sys/man Directory Structure

The source files for the *man* program are kept in section directories in $/sys/man. Each section is given its own directory, starting with the name ‘‘man.’’ and followed by the section name (e.g. man.LOCAL). The source files are similarly tagged with their section name (e.g. cat.C for *cat*, which resides in $/sys/man/man.C).

There is also an index file called *index* in $/sys/man. This index is a list of aliases for titles that do not have their own source files. For example, ‘‘rmdir’’ does not have its own source file, but is aliased to ‘‘rm’’ which includes documentation for ‘‘rmdir’’.

$/etc/rsrc/man.rsrc

The resource file for *man*, if it exists, contains a single line of all section names, separated by colons (:). The sections should be listed in the order in which titles should be searched. The default search order is:

```
ADM:C:S:NET:CP:CT:M:F:HW:LOCAL:G
```

Creating New Manual Entries

You can create new manual pages for utilities and scripts that you have developed. Formatting is accomplished by inserting dot macros into the manual entry file. For more information, refer to the manuals(F) entry.

Printer Output

To redirect output to the printer, use:

```
man -d -w 96 C * >.printer
```

This example enables dot-matrix formatting for a printer capable of displaying 96 columns of text. All titles in the ‘‘C’’ (Commands) section would be printed.

To print manuals to a PostScript device, such as a laser printer, see manps(CT).

File Redirection

To redirect manual output to a file, use >*filename* as the last argument.

Files

$/sys/man/* - directory of manual sections and entries,
$/sys/man/index - index to aliased titles.

See Also

help(C), manps(CT), manuals(F), whatis(CT)

Name

manps - Prints PostScript-formatted pages in the manual

Syntax

manps [*options*] [*section*] *title* ...

Description

Manps locates and generates pages for the named *title* from the designated *section* in the ProLine Reference Manual. The output is in the PostScript language format, and can be downloaded to any PostScript-compatible device (i.e. a laser printer or typesetting machine).

Since commands are given in lowercase, the *title* is always entered in lowercase. *Section* names are always uppercase. If no *section* is specified, the whole manual is searched for a matching *title*, and the first occurence found is printed. You can search through a group of sections by separating the section names with colons (:) on the command line.

Manps can generate pages for all entries of a given section by using an asterisk (*) as the *title*. A *section* argument must be given.

Options

The following *options* are recognized by *manps*:

-a	‘‘All’’ mode. Generates output for matching titles found in all sections in the search list.
-c	Generates a cover page.
-e	Even-up entry. An even page is generated for an entry consisting of an odd number of pages. The final page is blank, except for a header and footer. Use this option when generating manuals for two-sided printing where each entry will stand alone.
-E	Even-up section. An even page is generated after printing an odd number of pages in one or more entries (e.g. an entire section). Use this when printing without the -e option.
-l rows	Sets the maximum page length in rows.
-p	Progress mode. Prints the pathname of each title being processed.
-t title	Assigns a title to both the header and cover page (e.g. -t "Local Commands"). This is mostly useful when printing an entire section.
-w cols	Sets the maximum page width in columns.

File Redirection

Manps supports output redirection. To redirect manual output to a file, use *>filename* as the last argument. The output file can then be downloaded to a PostScript-compatible device.

$/etc/psprep

The PostScript output definitions are defined in $/etc/psprep. The default settings were chosen for maximum compatibility with most laser printers.

The default fonts used are Times Roman (12 pt.) for the standard text face, Courier (10 pt.) for monospaced displays, Helvetica (13 pt.) for subheadings and paragraph titles, and Helvetica (14 pt.) for headers and footers.

Files

$/etc/psprep - PostScript prep file,
$/sys/man/* - directory of manual sections and titles.

See Also

man(CT), manuals(F)

Name

whatis - Summarizes commands

Syntax

whatis *command*...
whatis -m [-q]

Description

Whatis looks up one or more *commands* in its database and gives the Name line taken from the manual section. The *man* command can be used to get more information. If * is given as the first argument, *whatis* displays the entire database.

The -m form causes *whatis* to make a new database. This is necessary whenever manual entries are added or removed. If the optional -q option is included, *whatis* is quiet while making the database -- no output is displayed.

Files

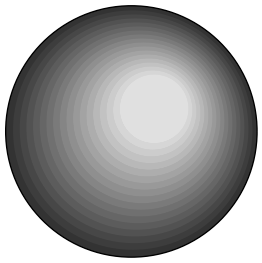
$/etc/help/whatis - database of manual headers.

See Also

help(C), man(CT), manps(CT)

ProLine

File Formats



Name

aliases - Description of mail alias files

Description

When the mail system is unable to recognize an address, it can search two alias databases for a valid address. Mail aliases come in two forms: user aliases and path aliases. A user alias is one which replaces the username in a mail address.

For example, if the system did not recognize the user ‘‘jsbach’’, it would look up jsbach in the user aliases database. If the system is instructed to send mail to pro-sample!user, and it does not talk directly to pro-sample, it would look it up in the paths database.

If the mailer can’t recognize a user or site name, and it can’t find a matching alias in either of the two databases, it will forward the message back to the sender with an error message.

User Aliases

The user alias database is the file $/etc/aliases. It contains lines in the form:

```
# This is a comment
root:           ~mdavis
sysop:          ~mdavis
mdss:           ~mdavis
postmaster:     ~mdavis
rnews:          ~rnews
null:           ~            # Mail just evaporates
news-update:    >/a/etc/news # Mail is written to a file
weather: >>/a/etc/wx  # Mail is appended to a file
bblue:          bblue@crash
friends:        danield, tom, bob, jay@snnark.uucp,
                jim@fpr.com, ddavis, jholt@adobe.com
my-group:       ~my.group
```

As shown, the username to match is followed by a colon, and the alias to replace that name follows. Multiple addresses in the alias can be given; this denotes a distribution list. If there are more names in a list than can fit on one line, extra lines can be included on subsequent indented lines, as shown in the friends entry.

Comments can be inserted by prefixing them with the pound-sign (#). Anything following the pound-sign up to the end of the line is ignored.

If a message was addressed to ‘‘bblue’’, it would be replaced with ‘‘bblue@crash’’ and passed back to the mailer for routing through the ‘‘crash’’ site.

If a local address in the list begins with a tilde (~), it forces sendmail to write the message into the named mailbox without verifying that the address is an actual user on the system. For example, the my-group entry is aliased to ~my.group. Any mail sent to my-group is added into the $/sys/mail/my.group mailbox, even though no user exists with that account name. If no mailbox names follows the tilde, letters to that alias are discarded.

If an address in the list begins with >, the letter is written into the pathname that follows, replacing it if it exists. If the address begins with >>, the letter is appended to the pathname that follows.

Path Aliases

The path alias database is the file $/etc/paths. It contains lines formatted in the same manner as user aliases. Example:

```
.cts.com: crash!*
alphalpha: pro-angmar!alphalpha!*
baron: pnet07!baron!*
cerf.net: pro-nbs!pro-fred!cerf.net!*
crash: crash!*
pro-aasgard: pro-aasgard!*
```

The sitename to match is followed by a colon, and the alias to replace that name is last. The asterisk shows where the remainder of the path should be placed into the address.

Comments can be included following the pound-sign (#).

If a message was addressed to ‘‘baron!jcurtis’’, the path alias entry would be substituted and the full address would expand to ‘‘pnet07!baron!jcurtis’’.

Notes

An alias can contain another alias, as long as they don’t cause an infinite loop to result. Nested aliases should be avoided, however, since it causes the mailer to work extra hard to resolve a route.

To speed up searches through these databases, copy them to your temporary directory ($tmpdir) if it is a RAM disk. Using *cp* commands in your $/etc/rc startup script is a good place to do this.

Files

$/etc/aliases - user alias database,
$/etc/paths - path alias database

See Also

domains(NET), forward(F), map(F), sendmail(C)

Name

cs - Conference System file formats

Description

Messages in the conference system are stored in a four-layer directory hierarchy:

```
                    CS Main Directory
                    /      |        \
          conference   conference    conference
          /   |   \        |           /    \
     topic topic topic   topic      topic   topic
       |      |    |       |          |       |
     msgs   msgs  msgs   msgs       msgs     msgs
```

The conference system structure begins with a main directory. This directory contains subdirectories for each conference on the system. In turn, these subdirectories contain subdirectories for each topic within the conference. A conference can have a maximum of six (6) topics. Inside of the topic directories are the individual messages.

CS Resource file

Cs requires a resource file, *$/etc/rsrc/cs.rsrc*, which contains the pathname to the CS Main Directory. The resource file also includes the text that is placed into the Organization field of Internet-bound posts. A typical resource file might look like this:

```
/a/cs
ProLine [pro-sol], Rancho San Diego, CA
```

The organization information should include the name of the host and its location.

Info Files

Each area in the CS structure may include a file called ‘‘info’’. Info files contain information relating to the area in which they exist.

For example, upon entering the conference system for the first time, the $/sys/pcs/info file is displayed. When joining a new conference, the info file that resides in the conference’s directory (along with its index and topic directories) is displayed, if it exists. Likewise, each topic directory can contain info files (along with the topic’s messages).

Conference Index Files

Each conference has an index file called ‘‘i..’’ in the conference directory. The index keeps track of the conference moderator’s name, the number of topics in the conference, and other information. The format of these files is given here, though you should never have to edit them directly. The *csmod* program handles any editing for you.

A typical index file might look like:

```
mdavis
1,4
news
1,0,7
vaporware:sewall@uconnvm.bitnet:1
0,1227,1327
groups:pro-news-groups@pro-sol;pro.news.groups:0
1,911,985
```

The first line contains the name of the moderator.

The second line, first number, is a flag: 1=open conference, 0=closed conference. The second number on that line is a count of the number of topics in this conference.

The lines that follow contain information about each topic. A topic has two lines of information. The first line is the topic name (i.e. "news"). If the topic is a networked topic, additional information is provided, separated by colon characters: the network address to which postings are sent, and a flag specifying if signatures are attached to network posts (0=no signatures, 1=attach signatures). The posting address field may also include the formal name of a newsgroup with a semicolon separating the address from the group name.

The line following the topic name contains three numbers:

(A) Flag: 1=read only, 0=read/write

(B) First active message number in the topic

(C) Next usable (unused) message number, this means that this number minus one is the last message that is stored on disk.

User Data Files

The conference system keeps track of the conferences each user is joined to. It also knows which messages a user has read. This information is stored in user's $/adm directory in a file named *csdata*.

The first line of a *cs* membership info file contains the conference name and, after a comma, the number of topics in that conference. The remaining lines specify the last message read plus 1 within each topic. If a user has resigned from that topic, that number is preceded by a hyphen. If a user resigns from a conference, the complete entry for that conference is removed including all the data for the topics within that conference.

Default Membership

If present, new users are automatically joined to the conferences listed in the

$/etc/default/csdata file. This file contains the name of each conference, followed by a comma and a zero, on each line. Example:

```
learn,0
proline,0
apple,0
```

Files

$/sys/pcs/cs.herald - Initial greeting upon entering *cs*,
$/sys/pcs/cs.list - list of conferences with descriptions,
$/etc/rsrc/cs.rsrc - CS resource file,
$/etc/default/csdata - list of default conferences users join,
csdir/*/i.. - index, status and config file for each conference,
$/adm/*/csdata - data files for each cs member,
$/etc/help/* - help files for CS commands.

See Also

cs(C), csmod(C)

Name

forward - Mail forwarding file

Description

Mail addressed to a local user may be automatically forwarded to any other e-mail address by creating a file called *forward* in the user's home directory ($/usr/username/forward). This file typically consists of one line which holds one e-mail address, but multiple lines and multiple e-mail addresses (space-separated) may also be given. Forwarding addresses can reference other local or offsite accounts.

For example, a user on this system named jsbach can create a *forward* file that contains this address:

```
johann@pro-musette.cts.com
```

Any mail destined for jsbach on this site will effectively be forwarded to johann@pro-musette.cts.com instead. Note that the local user jsbach will *not* receive a copy of the letter -- it is simply passed onto the forwarding address.

To forward a letter and have a copy retained in the local mailbox, include the user's name, a space (or a new line), and the forwarding address. Example:

```
jsbach
johann@pro-musette.cts.com
```

Mail addressed to jsbach is copied to jsbach's local mailbox as well as forward to johann@pro-musette.cts.com. A number of additional forwarding addresses can be supplied.

Note

To discontinue forwarding, remove or rename the *forward* file.

For loop protection, an address in the forward file cannot reference another local user that has a *forward* file.

Files

$/usr/*/forward

See Also

aliases(F), mail(C), sendmail(C)

Name

manuals - Format of manuals

Description

The ProLine Reference Manual consists of individual files, called *entries*, that describe a part of the system, be it a command, a file format, miscellaneous information, etc.

Manuals follow a specific visual format. Shown at the top of each page is an entry's title and section, e.g. MANUALS(F), and at the bottom is its modification date, page number, and date printed.

Subheaders

These subheaders are standard in manual entries and typically follow in the order presented here:

Name	A short, one-line summary of the document follows.
Syntax	Included for entries that describe a command with parameter arguments.
Description	Begins the descriptive body of the entry. The body of an entry is generally undefined. Though, it should not use any subheader other than those described here. It may contain, however, its own paragraph headings.
Note	(or Notes, Bugs, Warning, Important, etc.) This subheader proceeds important information that should stand out from the body of the description, such as hazards or side effects.
Diagnostics	Error messages and their meanings.
Files	If any files are associated with an entry, they are listed after this subheader, usually near the end of the entry. Each file listed should include a short description.
See Also	Usually, an entry ends with a ‘‘See Also’’ subheader, followed by a reference list to related entries.
Author	(Optional). The name of the author (for the entry and/or the command it describes) is given, along with the author's e-mail address. If this subheader is not included, it is assumed that the author is Morgan Davis.

Internal Structure

Manuals are punctuated with ‘‘dot’’ commands for interpretation by processing programs such as *man* and *manps*. Each command has the form of a period (.) followed by a two-letter command name. Arguments to a command are separated from the command by a space. Text that does not start with a period is considered text to be processed. Text portions need not be formatted in the source file, as the processing programs will perform all the necessary formatting.

Dot Commands

The ‘‘dot’’ commands are described below. In the descriptions, ‘‘s’’ denotes a set of characters while ‘‘n’’ denotes a number. Arguments shown between [ ] brackets are optional. If the optional number is omitted, zero is assumed, unless otherwise noted.

.TH s1 s2	Title Header. *s1* is the entry’s title, followed by a space, and then the associated section, *s2*. Both arguments must be in uppercase. (e.g. .TH MANUALS F)
.DA s	Date. *s* contains the date of the last modification of the entry in this format: 12 March 1992.
.SH s	Subheader. Displays *s* fully left justified.
.LM [n]	Left Margin. The value of *n* is added to the current left margin. A negative value narrows the margin, while a positive value widens it.
.RM [n]	Right Margin. The value of *n* is added to the current right margin. A negative value narrows the margin, while a positive value widens it.
.PP [s]	New Paragraph (with optional *s* paragraph header).
.IP [n [s]]	Indented Paragraph. Begins a new paragraph, setting the indentation to *n* columns. If *s* is included, it is displayed at the left margin, and the print position tabs out to the indentation point. The effect is what you see here -- the dot commands are left-margin justified, while the descriptions are indented. Use .IP with no arguments to disable indentation.
.CN s	Centers *s*, followed by line break.
.SP [n]	Spaces. Print *n* spaces (default is 1).
.RC [n] s	Repeat Character. Prints the *s* character *n* times. If *n* is omitted, *s* is repeated up to the right margin and a line break follows. Use this command to display characters that would be misinterpreted if included in text (e.g. ‘‘.RC 1 _’’).
.BR [n]	Break. Inserts *n* line breaks (default is 1). This moves the printing position to the next line, respecting current indentation settings.
.AF [n]	Auto-Fill. If *n* is given, auto-fill is turned on (1) or off (0). If no argument is given auto-fill is toggled. Proceeding text lines are displayed with no formatting. Blank lines are ignored, but commands

	such as .BR are still recognized.
.UC [s]	Underline Character. Sets the emphasis-toggle character to *s* (default is the underscore ‘_’). Text selections surrounded by this character are emphasized. For video displays, this results in either highlighted or underscored text. For dot matrix printers, the text is underlined. For PostScript devices, the text is italicized.
.NP	New Page. Ends the current page by printing the appropriate number of lines, followed by the page footer.
.IF n	If fewer than *n* lines remain on the page, a new page (.NP) is performed. Include .IF before tables or lists that should not be broken across pages. The .SH, .PP and .IP commands include built-in widow control, as if ‘‘.IF 2’’ precede them.
.TB n	Tab to column *n* from current print position.

An excellent way to learn to ‘‘program’’ a new manual entry is to compare the contents of an existing entry with its printed output, noting how each dot command effects the display.

Typographic Conventions

Whenever reference is made to the command being described by the entry, always highlight it using the underscore/highlight quoting character. Similarly, reference to other commands should be highlighted as well. Highlighting is valuable for making urgent or important passages of text stand out. Such text is displayed in italics on PostScript printers.

Use typographer’s quotation marks. These consist of the apostrophe (’) and accent grave (‘) characters, either singularly or in pairs. ‘‘Like this,’’ for example.

Whenever examples are used that reference items that the user might see on the screen, turn off the auto-fill mode (.AF). Text that is not auto-filled is displayed in Courier, a monospace font, on PostScript printers. This has a the desirable effect of making the text stand out in a ‘‘computer-like’’ typeface.

Files

$/sys/man/* - contains manual sections and entries.

See Also

man(CT), manps(CT), whatis(C)

Name

map - Networking map file

Description

Map files are used to catalog and identify ProLine systems in our network. Each ProLine site must create its own map entry file and submit it for processing, or else the rest of the network will not know the site exists.

In the event that paths databases are automatically generated as a result of processing map files, network sites depending on automatic routing and smart mailers to handle delivery require your map entry to be accurate. Inaccurate data will, at the least, cause delayed or more expensive deliveries, and at the most, failed deliveries.

Building Your Map

You’ll need to create your map entry file in any text editor. All lines beginning with ‘#’ are comment lines to *pathalias*, however the UUCP Project has defined a set of these comment lines to have a specific format so that a complete database can be built.

Each comment field in your map entry file uses this format:

```
#<field id letter><tab><field data>
```

Note that <tab> means the actual Control-I character (ASCII 9), or six spaces to put you into the next tab column.

The entry should begin with the #N line, and end with a blank line after the routing data. The final blank line is important in that it denotes the end of your routing list. Here’s a template:

```
#N      sitename
#S      Machine type; operating system
#O      Organization name
#C      Contact person’s name
#E      Contact person’s electronic mail address
#T      Contact person’s telephone number
#P      Organization’s address
#L      Latitude / Longitude
#R      Remarks
#U      Netnews neighbors
#W      Who last edited the entry; Date edited
#
sitename        remote1(FREQUENCY), remote2(FREQUENCY),
                remote3(FREQUENCY)
```

(one blank line goes here!)

Example of a completed entry:

```
#N      pro-sol
#S      Apple IIGS; ProLine 2.0
#O      Morgan Davis Group
#C      Morgan Davis
#E      mdavis@pro-sol.cts.com
#T      +1 619 670 0563
#P      10079 Nuerto Lane, Spring Valley, CA 91977-7132
#L      32 47 N / 116 56 W
#R      24 hour operation at +1 619 670 5379
#R      300 to 9600 bps V.32, V.42, V.42bis
#W      pro-sol!mdavis (Morgan Davis); Wed Jun 15 18:00:00 PDT 1988
#U
#
pro-sol pnet01(HOURLY-5), pro-mercury(HOURLY), pro-mars(HOURLY-5),
        pro-sat(HOURLY), pro-simasd(HOURLY-5), pro-test(HOURLY),
        pro-carolina(EVENING), pro-exchange(WEEKLY/2), pro-party(WEEKLY),
        pro-angmar(WEEKLY/2), pro-charlotte(WEEKLY), pro-colony(WEEKLY),
        pro-pac(WEEKLY), rti-austin(DAILY), pro-hobbyist(DAILY),
        pnet08(DAILY*2), pro-avalon(HOURLY), pro-vide(HOURLY+5),
        pro-starbase(DAILY), pro-ascii(HOURLY), pro-la(WEEKLY)
```

Be sure to include that final blank line at the end of your map file, because maps are submitted through the mail system which can attach signature files to the end of your messages.

Map Fields

The following is a description of each field in the map entry. Each field prefix should be present in your map, even if the field is blank.

#N Your system's name. One of our goals is to keep duplicate host names from appearing because there exist mailers in the world which assume no duplicates (and attempt UUCP path optimization on that basis), and it's just plain confusing to have two different sites with the same name.

#S This is a quick description of your equipment (i.e., ‘‘Apple IIGS’’), followed by the version of ProLine you're running.

#O This should be the name of your organization (company, school, computer club, etc.), squeezed to fit inside 80 columns as necessary.

#C The full name (or names, separated by commas) of the person responsible for handling queries from the outside world about your machine.

#E This should be just a machine name, and a user name, like ‘‘pro-sol!mdavis’’. The site name does not have to be the same as the #N field (i.e. the contact ‘‘lives’’ on another machine at your site).

Multiple electronic addresses should be separated by commas, and all of them should be specified in the manner described above.

#T Telephone number of the contact person in this format:

+1 415 642 1024

This is the international format for the representation of phone numbers. The country code for the United States of America (and Canada) is 1. Other country codes should be listed in your telephone book. If an extension is needed, use this format:

+1 415 549 3854 x37

Multiple phone numbers should be separated by commas, and all of them should be completely specified as described above to prevent confusion.

#P Postal address of the contact person. Include all information necessary for anyone in the world to reach the address through paper mail.

#L The site's latitude and longitude (or nearest approximation). This should be in the following format (two fields, with optional third):

DD MM [SS] N|S / DDD MM [SS] E|W [‘‘city’’]

The first number is Latitude in degrees (DD), minutes (MM), and seconds (SS), and a N or S to indicate North or South of the Equator.

A slash separator.

Second number is Longitude in degrees (DDD), minutes (MM), and seconds (SS), and a E or W to indicate East or West of the Prime Meridian in Greenwich, England.

Seconds are optional, but it is worth noting that the more accurate you are, the more accurate maps we can make of the network (including blow-ups of various high density areas, like New Jersey, or the San Francisco Bay Area).

If you give the coordinates for your city (i.e. without fudging for where you are relative to that), add the word ‘‘city’’ at the end of the end of the specification, to indicate that.

If you're not sure of your coordinates, call any land surveying company in your area, or your local government offices.

#R Remarks. As noted before, all lines beginning with a ‘#’ character are comment lines, so if you need more than one line to tell us something about your site, do so before the other fields.

#U Network news neighbors. The Internet is the network that moves netnews around. Unless you receive news directly from an Internet site, leave this blank.

#W Who last edited the map entry and when. This field contains an e-mail address, a name in parentheses, followed by a semi-colon, and the output of the UNIX date program. Example:

```
pro-sol!mdavis (Morgan Davis); Sat Feb 23 05:03:00 PST 1988
```

The rules that apply in the contact's e-mail address apply here also. (i.e. only one system name, and user name). This field is used for automatic aging of the map entries.

The Connectivity List

After all the '#' fields comes a list of systems that your site talks to directly. This list begins at the left margin with the name of your site, followed by the remote systems yours talks to. Commas are used to separate each site in the list, and any sites that are positioned on following lines must begin with one or more tabs. Example:

```
pro-test        pnet01(DIRECT), pro-sol(HOURLY),
                pro-lumen(DAILY/4)
```

Immediately following the LAST entry in this list is a completely blank line.

The items in parenthesis which immediately follow a site name are imaginary cost values associated with that connection. Cost values are measured in both frequency of polling and/or answering times, in addition to other factors (such as long distance charges, modem speeds, etc.). *Pathalias* uses these values to calculate the lowest cost routes from site to site.

The cost breakdown is:

DIRECT	200	Local call, many connections per hour
DEMAND	300	Toll call, many connections per hour
HOURLY	500	One connection per hour
EVENING	1800	Multiple calls during low-rate hours
POLLED	5000	One or more polls per day by remote site
DAILY	5000	One connection per day
WEEKLY	30000	One connection per week
DEAD	(high)	Temporarily inoperable connection

If a site goes down for good, or you decide to discontinue connections with one, remove the site and its cost entirely from your map. A DEAD cost means that you're still connected and you accumulate mail for that site.

Incrementals of these costs can be used, too. For example, DAILY*2 would mean every other day. DAILY/2 could mean twice a day or twice an evening if EVENING isn't quite the right description of the connection.

Steps For Determining Costs

Due to the nature of the ProLine software, DIRECT and DEMAND are impossible costs to use. These indicate a connection where your site would immediately poll another upon receipt of mail for that site. ProLine polls on a schedule, not because of the immediate presence of outgoing mail. Do not use these costs.

The best a ProLine can do is HOURLY, and that is only possible if a site makes two or more attempts to poll another site in the course of an hour. This is because ProLine can only poll when it is not busy doing something else (system maintenance or hosting a user). It realistically cannot connect once an hour unless it makes multiple attempts (every 15 to 30 minutes). If a sites a single attempt each hour to connect with another site, the cost is probably more accurate as HOURLY*3 (meaning, every three hours).

1. Use the above values to establish a starting point for your connectivity, and base it only on the sites that you call.

2. Next, for each site that calls you, estimate the costs based on the frequency of calls you receive. If the estimated cost of incoming calls is *lower* than your initial cost, use the lower value. If it is the *same* or *higher*, make no changes. Do not add or subtract values based on two-way communications. If the site that calls you is one you don't call, use POLLED.

3. At this point, you should see that your calls are the primary cost factor, and calls you receive might lower cost, but will never increase it.

Fudge Factors

You can include values to raise or lower a cost to show preference or avoidance. Typical values might be a -10 preference for a high speed connection, or a +5 or +10 to avoid a certain site for which there might be several parallel paths. Example: pro-site(HOURLY+10).

You can also declare a high cost to a site if you don't want net traffic for that site going through your machine. Such an entry will not effect your local use of that connection.

Storing Your Map Entry File

After you create your map file, place a copy of it in the $/sys/maps directory using the name of your site as the filename in legal ProDOS format. For example, ‘‘$/sys/maps/pro.sol’’.

At the present, $/sys/maps is used only for local map entry storage. But as tools become available for building and updating maps right on a ProLine system, this directory will become more important.

Submitting Your Map Updates

After creating your map entry, you must submit it to the map coordinator for archiving and potential processing. Using the electronic mail system, send your map to:

mapcoord@pro-sol

(If you use *rcp* to send it, be sure to specify ASCII text format with the -a option).

Currently, map submissions are archived on pro-sol.cts.com (available via mail server request), and various site list maintainers.

Files

$/sys/maps/* - map storage area.

See Also

aliases(F), intro(NET)

Name

plush - Menu format and programming

Description

The $/etc/plush.m text file contains specially-formatted menu definitions that can be modified to alter the way *plush* presents commands to the user. The exact formatting of this file is critical -- a mistake will cause incorrect operation which may not be reported. Since the file contains text commands, any text editor can be used to edit it.

The Menu Definition File consists of individual structures, one for each menu. Multiple menus are separated by blank lines. The definition file is terminated by the ‘‘END’’ command. The typical layout of this file would be:

```
Menu #1 Structure
<a blank line>
Menu #2 Structure
<a blank line>
Menu #N Structure
END
```

Note that there is no blank line between the last structure and ‘‘END’’.

Format of a Menu Structure

Each menu structure consists of three items: a menu title, the menu’s prompt, and the menu’s key-command list:

```
Menu Title
Menu Prompt
<First key-command line>
<Second key-command line>
...
<Last key-command line>
<Key-command list terminator>
```

Here is an actual menu structure for a "Games Menu":

```
Games Menu:
Games:
3,2,0,Ll,L,L - List all games,DO ls $/games
3,2,0,Pp,P,P - Play a game,IN Which game;DO $/games/$IN
3,2,0,,,,
3,2,0,Ff,F,F - Fortune cookie,DO $/games/fortune
3,2,0,Oo,O,O - Other fun things,>> Other Fun Things
9,9,9,,,,
```

The first line contains the name of the menu (Games Menu).

The second line is the menu’s prompt string (Games:). If the prompt begins with a < (less-than) character it identifies a menu as an *extension* menu. Extension menus allow the user to make one selection, and then the menu is closed. Normally, once a menu is opened it remains open until the user manually closes it by pressing a ‘‘backup’’ key.

Beginning with the third line is the key-command list (key-command lines are discussed in detail next). This list contains seven fields that reside completely on one line of text, though the line can be longer than the screen’s width.

Key-Command Lines

Each key-command line contains seven fields, separated by commas. Fields are presented in the following order (using a sample line as reference):

```
3,2,0,Ll,L,L - List all games,DO ls $/games
```

Group-ID:	(3) Determines if this line is accessible based on the user’s group-ID. Groups are: (0) root or sysop, (1) staff, (3) guest. If the group-ID field contains a number which is less than the user’s group-ID, the command is not available. This can allow certain commands to be inaccessible for lower-level users.
Location:	(2) This code, 0 through 2, determines if the selection is available based on the user’s current location: (0) the local console, (1) remote terminal, or (2) either local or remote.
Priority:	(0) Priority is: (0) standard, or (1) super-user override. This allows *plush* to perform certain commands that would otherwise be illegal due to the user’s group-ID permissions. As an example, a guest or staff level user cannot view a text file in the $/etc directory. But if root-user priority (1) is specified, such access is permitted. WARNING: Giving root-level access to some commands could allow a non-root user to damage your system!
Keys:	(Ll) Two letters or symbols, or blank. These characters define the keystrokes that invoke the command for this line. Usually, upper and lowercase versions are used. "+=" would be used so that pressing the "+" key would not require the user to hold down SHIFT. If this field is left blank, this item is not selectable but the description is displayed in the command list. This can be used to create blank lines or special separators.
Response:	(L) This letter, or string of characters, is echoed back to the user after pressing one of the two keys as long as the "hot-key" input mode is used.

Description:	(L - List all games) Describes the command. Shown when the command list is displayed. A blank line is displayed if the response field is blank. However, if the response field contains characters and the description is blank, the menu item is treated like any other, only it is not displayed. This is useful for creating hidden menu items.
Command:	(DO ls $/games) The command line. Commands start with a two-letter code followed by arguments (if any). Multiple commands can be placed in this line by separating them with a semicolon (;).

The key-command list is terminated by a key-command line that consists of a group-ID of 9, location of 9, priority of 9, and the remaining fields all blank. Example:

```
9,9,9,,,,
```

Command Codes

Plush interprets the following two-letter commands (uppercase required):

DO	Launches an external program. *Plush* searches $/bin, $/sys/bin, $home/bin, and $/sys/local/bin to find the program. If it does not reside in one of those areas, include the complete path to the program. Use $/ to specify the ProLine System Prefix. The DO command name is optional, ‘‘DO df’’ and ‘‘df’’ will produce the same results.
ED	Edits a file. The full path to the editable text file must be given.
VU	Views the text file given as the argument.
>>	Opens the submenu whose name is given as the argument. It is best to place the named menu AFTER the menu in which this command is issued. This is because *plush* first searches for menus in a forward direction (from the current menu down to the last menu). If the menu can’t be found, *plush* searches from the current menu backwards to the first menu in an attempt to locate it.
<<	Closes the current menu and returns to the previous one.
EX	Exits *plush*. This is allowed if *plush* is invoked from *csh* or some other shell.
BY	Bye. Logs the user out, disconnecting the modem.
PR	Prints a message. If no argument is given a blank line is printed.
CD	Changes directories to the argument given. If no directory is specified, the current working directory becomes the user’s home directory.

RM	Removes the named file.
IN	Accepts a line of input, storing it in a special variable called $IN.
QY	Query-yes. Prompts the user with a Yes/No question, Yes being the default. If the user's response is Yes, any remaining commands in the command line are performed. If a negative response is given, the command line is immediately terminated.
QN	Query-no. Same as QY, except the default is No.
CL	Clears the screen.
TB	Test blank. Checks the argument that follows to see if it evaluates to nothing, a blank value. For example, ‘‘TB $IN’’ cancels the command line if the last user input was blank (nothing entered).
TF	Test file. Checks the file argument that follows to see if it exists. For example, ‘‘TF $IN’’, cancels the command line if the file name in $IN is not found.

Special Characters

It is very important that no illegal characters such as commas, colons or misplaced quotation marks are used within any of the seven fields of the key-command line. Commas and colons can be used as long as the entire field is enclosed in quotes. For example:

```
3,2,0,Bb,B,"B - Bye: disconnect, hangup",BY
```

To invoke a command with a single argument that consists of spaces, enclose the argument in single-quotes (’). Example:

```
0,2,0,Mm,M,M = MDSS Account,adduser -m ’bin/mdss -x BLX’
```

Plush supports the standard C-Shell escape (\) and control-code (^) prefix characters. Any tilde (~) character is replaced with the path to the user's home directory.

Variables

Plush recognizes the following variables:

$IN	Input from the most recent IN command.
$/	Path to the ProLine System Directory.
$spool	Path to the spool directory.
$tmpdir	Path to the temporary files directory.

Notes

Plush supports 100 menus, and each menu can include 30 items. This limitation exceeds the capacity of most of the system's text editors, so working with a large plush.m file may require utilizing an external text editor or word processor.

Since *plush* works with only one menu at a time, it operates faster if the menu file is located on a fast disk (e.g. a RAM disk). To utilize the $tmpdir option, include a copy command in the $/etc/rc file to put a copy of plush.m into $tmpdir.

See plush(C) for details on where *plush* looks for its menu files.

Files

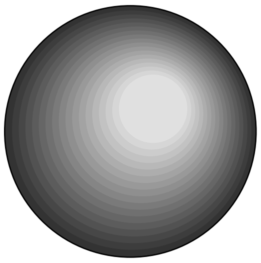
$/adm/*/environs - user's environment settings,
$tmpdir/plush.env - temporary environment file.

See Also

csh(C), plush(C), setenv(C)

ProLine

Games



Name

af - Add a fortune to the fortunes file

Syntax

af [-o]

Description

Af, a shell script, adds a fortune to the end of a fortune data base. The -o flag causes *af* to use the optional fortune database, one traditionally containing offensive fortunes.

Af asks you to enter the entire fortune, and then press RETURN. If the fortune is longer than the display width, *fortune* correctly wraps it at word breaks when displayed.

Af allows control characters to be included by using the caret (^) notation. However, Control-M (RETURN), must never be used. If a newline is desired in the middle of a fortune, Control-N (^N) will be recognized as such by the fortune program.

If a non-root user submits a fortune with *af*, it is not actually appended to the database, but rather is sent to the system administrator through the mail system for review. The administrator may decide whether to add it manually.

Files

$/games/lib/fortunes - standard fortune database,
$/games/lib/fortunes2 - alternate fortune database.

See Also

fortune(G)

Name

bank - Simulated banking system

Syntax

bank [*options*]

Description

In games where money is the reward, it is desirable to maintain a balance of cash between various games and be able to utilize your earnings. The ProLine Users Bank provides this service, and helps regulate a simulated economy inside this system.

Your bank account allows you to make cash deposits and withdrawals with your savings account, which can earn interest. It keeps track of how much money you are carrying with you (cash-on-hand). You can transfer funds to another user on the system. You can take out loans and pay them back as you earn money by playing (and winning) games. There are no service charges or maintenance fees.

Your account is created automatically (with a gift balance) the first time you invoke it. For more details about using the *bank* interactively, issue its built-in <H>elp command.

Options:

-a n	Add cash. This option adds cash (value *n*) to the current user's cash on hand. The current user defaults to the person issuing this command. (See -*u* or -*g* for setting the current user to a particular account).
-c n	Change cash. This sets the cash on hand to the specified value *n* for the current user.
-g game	Set game user. This sets the current bank account to the one owned by the specified *game*. Games like *blackjack* can maintain bank accounts just as users do.
-i	Interactive. Normally, when invoking *bank* without arguments, the user enters an interactive mode. When options are used, the functions they perform are completed and the user returns to the shell. This option is used in conjunction with other options to send the user into interactive mode. Typically, this is used after setting the current user with -*u* or -*g* to work with their account interactively.
-s n	Subtract cash. This subtracts from cash on hand the specified value *n* for the current user.
-q	Quiet. Some options, such as setting the current user, issue feedback. This option disables such feedback.
-u user	Set user. Sets the current account to the one owned by *user*.

-w Net worth report. Quick and easy way to find out what you’re worth.

Of all these options, only -*w* can be used by non-Super Users.

Setting Up Accounts

As stated, user accounts are created automatically when users enter the *bank*. Accounts for games, like *blackjack*, need to be created so that the game (casino) has cash with which to work. This is usually done like so:

```
bank -g blackjack -c 50000
```

This sets the current account to the *blackjack* game (hence -*g* option as opposed to -*u*), then assigns $50,000 cash to the account. Game accounts, like the users who play the games, operate out of their pool of cash on hand.

Note

The order of options is generally significant. That is, changes invoked by options occur in the order in which they’re presented. Thus, the following reversal of options can have undesirable affects:

```
bank -s 100.25 -u melvin
```

This subtracts $100.25 from the current account (yours!), then sets the current user to ‘‘melvin’’. This was probably not intended. Consider this:

```
bank -w -u melvin -w
```

This reports *your* net worth, sets the user to ‘‘melvin’’ and then reports his net worth.

Files

$/games/lib/bank.help - online help file,
$/adm/*/bank.account - user’s bank account file,
$/games/lib/*.acct - game bank account files.

Author

Morgan Davis (mdavis@mdg.cts.com)

Name

blackjack - Blackjack card game

Syntax

blackjack

Description

In *Blackjack*, the object of the game is to beat the dealer by holding a hand of cards that total more than the dealer's without going over 21. The values of the cards are: Ace, 1 or 11, as the holder wishes; king, queen, jack, ten, 10 each; any other card, its number. An ace and face card or ten in the first two cards are called blackjack.

Starting:

The shoe contains two regular decks of cards and is shuffled upon approaching the table. Before cards are dealt, the player places a bet ($5 or more but no more than $1000). The initial cards are dealt in this order: dealer-player-dealer-player. The first two cards are face down, and the remaining cards are always dealt face up.

Player Options:

After the initial deal, the player may choose the following:

<H>it	to request an additional card.
<S>tay	to stop and play the hand being held.
<D>ouble	to double the wager and take only one more card. This is allowed only when holding two cards, and is typically done when the player is confident that the next card will beat the dealer. Spli<T> to split a hand of two identical cards (except for aces). The player has a chance to play two separate hands and use the best one against the dealer.

Dealer:

The dealer turns up the face-down card and continues to draw cards as long as his total is 16 or less and must stand when his total reaches 17 or more.

Any time the bottom of the deck is reached, the dealer reshuffles without interrupting game play.

Settlement:

The winner is whoever stands with the highest total not over 21. In the case of a tie, no winner prevails and it is considered a stand-off. The loser pays the winner the amount of the wager, and in the case of the player (not the dealer), a bonus may be awarded as follows:

For blackjack, 1 1/2 times the amount of the bet.

For 21 or less in five cards, double; in six cards, triple.

For 21 composed of three sevens, triple; composed of 6-7-8, double.

Scoring:

Players who leave the game with more money than when they arrived are eligible for placement on the high score list. The listing shows the top ten players who have the largest winnings in a single session, and shows the number of hands won, tied, and lost.

Note: Earnings or losses are registered with the ProLine Users Bank and affect a player's cash-on-hand.

Super-User Installation

Use the *bank* to create a bank account for the *blackjack* casino. It cannot be played until an account is created.

The amount assigned to the game should be fairly significant if a lot of gamblers are expected to do well at the tables, otherwise the *blackjack* account may go broke. It is fun to watch the game's account balance to see if the house is winning over the players.

For a real thrill, have *blackjack* start out by securing a loan from the *bank*, instead of just assigning an arbitrary amount of cash. Then pay the bank back as the *blackjack* tables turn a profit. To do this, invoke *bank* as follows:

```
bank -g blackjack -i
```

This sets the current account to the *blackjack* game, then enters interactive mode where you can visit the loan department.

Files

$/games/lib/blackjack.help - online help file,
$/games/lib/blackjack.top10 - high score history,
$/games/lib/blackjack.acct - game bank account file.

See Also

bank(G)

Author

Morgan Davis (mdavis@mdg.cts.com)

Name

cyber - Cyber World interactive fiction adventure

Syntax

cyber

Description

A rip in the fabric of time lands you into the future following a devastating bio-war. Only Earth’s buildings, computers, and robots survived. Can you escape alive?

Exploration in Cyber World is like most adventure games: enter simple one or two-word commands to move, manipulate objects, and examine things you encounter. For example, ‘‘examine coins’’, ‘‘attack robot’’, ‘‘look’’, and so on. All objects have helpful descriptions -- it’s a good idea to look at everything for clues and avoiding hazards.

Move about using standard compass directions, abbreviated if you prefer (e.g., ‘‘n’’ is equivalent to ‘‘go north’’). All exits are clearly marked.

If a robot is present, you can get assistance from it by calling or talking to it. Warning: Some robots possess a sinister demeanor and may be dangerous to your existence!

Use ‘‘save’’ to preserve your current session in case you have to quit early, or want to have a backup position. Use ‘‘restore’’ to resume a saved session where you left off.

Use ‘‘help’’ for a list of commands and abbreviations.

History

Cyber World is a major revision of Daniel Tobias’ ‘‘Planet of the Robots’’ written in 40-column, uppercase Applesoft BASIC in November, 1981. The game originally occupied a 25K BASIC program file, which didn’t leave any room for expansion. Descriptions were, needless to say, not verbose. The full enjoyment of the plot could not be realized.

In 1993, the game was obtained by Morgan Davis who converted it to MD-BASIC, reformatting all the internal messages for standard 80 column displays, and moved the vocabulary, room and item descriptions out of the game and into an external data file. This allowed descriptions to be rewritten and expanded, plus made it possible to add large visual descriptions to items, such as the Deep Thought computer screen, a map for the mall, and so on. New puzzles and nuances to the plot were added to make the game richer with more interactive fiction. The resulting program slimmed down to just 10K but it gained 40K of external data.

The only significant change to the original game was the removal of a built-in low-res graphic arcade game in which the player engaged from within the mall’s arcade. This omission allowed the game to be played over a standard text terminal (e.g., BBS). As a consolation, the arcade game turned into something more like a slot machine that can pay back double the cost of playing it (if you’re lucky).

Files

$/games/lib/cyber.data - data file,
$/adm/*/cyber.save - saved session file.

Author

Morgan Davis (mdavis@mdg.cts.com), Daniel Tobias

Name

fortune - Print a random, hopefully interesting, adage

Syntax

fortune [-o]

Description

Fortune with no arguments prints out a random adage, not unlike that found in a fortune cookie. The -o flag causes *fortune* to randomly select a fortune from either the main or alternate list of adages, often used for potentially offensive ones.

Files

$/games/lib/fortunes - standard fortune database,
$/games/lib/fortunes2 - alternate fortune database.

See Also

af(G)

Name

today - Events on this day throughout history

Syntax

today [-s]

Description

Today prints famous birthdays and historical events for the current date in history. Using the optional -*s* flag (suspend redundancy) causes *today* to print the historical data for you only once per day.

Adding New Dates

You can add your own dates to the database by editing the files in $/games/lib/today. Here's an example for September, found in the file $/games/lib/today/sep:

```
B09011875 Edgar Rice Burroughs, novelist, Ah-ee-ah-ee-ah!
B09021838 Queen Liliuokalani (last queen of Hawaii).
B09021952 Jimmy Connors, tennis brat
S09011939 Germany invades Poland, starts World War II.
S09011952 Sutro Baths purchased by George Whitney.
S0901    2Labor Day, a legal holiday
S09021620 The Mayflower sets sail from Plymouth with
S09021620C102 Pilgrims.
```

Explanation

Column 1	The record type which can be a B for birthdays or S for special dates in history.
Cols 2-5	Month and day as MMDD with leading zeros.
Cols 6-9	The full year of the event as four digits. If left blank, the current year is used.
Column 10	A special processing flag. Values may be: BLANK for no special processing. C to continue a message from the previous line. Note that you MUST repeat the date fields. DAY-OF-WEEK digit where 1=Sunday ... 7=Saturday. This will cause the message to display only if the date falls on a certain day of the week.
Cols 11-70	The message to display for this date.

Files

$/games/lib/today/* - databases for each month of the year.

Authors

The ProLine version was written by Jeff Jungblut (jeff@pro-avalon.cts.com). The original database is from a similar public domain program for the IBM PC called TODAY/PC.

ProLine

Miscellaneous

Name

scripts - All about shell scripts

Description

This entry is an addendum to the C-Shell (*csh*) entry. Refer to *csh* first for introductory details on shell script files.

Shell scripts are text files that contain commands to be executed as a batch operation by the shell. Anything that can be performed at the shell prompt can be executed from a shell script. Here's a sample:

```
# First, set up some command aliases
alias   ls      ls -F
alias   c       "clear; ctime"
alias   send    sx -yl
alias   recv    rx -yl

# Now print out a random, hopefully humorous adage
fortune

mail -n         # read new mail (if any)

# Lastly, set up some variables
set bob=$/usr/bob
set prompt="\$time[4]> "
```

A file such as this can be created using a text editor and saved in your user directory. You can execute it from the shell just by typing ‘‘source file’’, where ‘‘file’’ is the name of the script you just created.

You can avoid having to enter the shell's ‘‘source’’ command to run a shell script by changing the filetype attribute of the script to ‘‘cmd’’. To change a text file to an executable ‘‘cmd’’ (command) file, use the *setfile* command. If you've just created a script and named it ‘‘wonka’’, for example, the command:

```
setfile -t cmd wonka
```

makes it executable. To run it, you simply enter its name. Note that the editor can edit ‘‘cmd’’ files as if they were ordinary text files, so using *setfile* is only necessary once after the file is first created (editing the file does not change its type). If you plan on creating many shell scripts, they're typically stored in a directory called ‘‘bin’’ in your $home directory (e.g. $home/bin/wonka).

There is a special ‘‘cmd’’ file named ‘‘login’’ in your directory which is automatically executed after logging in. You can use an editor to tailor it to your heart's desire, making your session on ProLine a unique experience.

Special Script Commands

Your scripts have the ability to use some special commands that allow for greater flexibility in the way they execute:

exit	halt a running script and return control back to the shell for regular keyboard-based command input. The *exit* command accepts a numeric argument as a result code to pass back to the shell or calling process. Any non-zero result is considered an error. If no argument is given, zero is assumed (no error).
if	test for a certain condition, such as the existence of a file. If the condition is true, script execution continues with the commands which follow until an *else* or *endif* command is encountered. If the condition is false, the first *else* or *endif* command on a new line is searched for, and execution will continue with the commands that follow it. (See *if* for complete details on if, else, and endif)
read var	read a line of input from the user and assign it to a shell variable.

You can place comments into your scripts by putting a pound-sign (#) before each comment. The shell ignores anything that follows.

Scripts can also make use of any arguments that are entered at the command line when the script was launched. This is done by making reference to the argc and argv shell variables which hold the argument count and argument list respectively.

Consider following script called ‘‘view’’:

```
if $argc < 2 then
     echo -n "Name of file to view? "
     read file
else
     set file=$argv[1]
endif

if ! -f $file then
     echo Can’t seem to find $file!
     exit -1
endif
echo ------------------
cat $file
echo ------------------
```

View begins by checking the argument count in argc. Since the command itself is part of the argument list, the argument count is always one plus the number of arguments following the command.

If $argc is less than two, meaning it must be one because only the name of the script was entered, then it prompts the user to enter a file name. Otherwise, if $argc is two or more, it assigns the file variable to the first argument, $argv[1].

Next, the view script tests for the existence of the file. The ‘‘!’’ operator is a logical NOT, which reverses the logic. Thus, the statement reads, ‘‘if the file does not exist’’. Should the file be non-existent, an error message is printed, and the script exits with -1, which sets the shell’s status variable to -1.

If the file does exist, it is displayed, bordered by lines of dashes.

Ignoring Errors

Normally, if an error occurs (due to a command or process that returns a non-zero result), a running script is halted. To avoid this, unset the exit shell variable. To restore normal error handling, set exit (no assignment value is necessary -- it just has to exist).

Note

A script can call another script as a command, but when the second script stops, the first one WILL NOT resume execution. To have execution of the calling script resume, launch the second script through a new shell. Example:

```
# This is the first script
$shell source script2       # invoke 2nd script via a new shell
echo It worked              # when the 2nd is done, it returns here
# end of first script
```

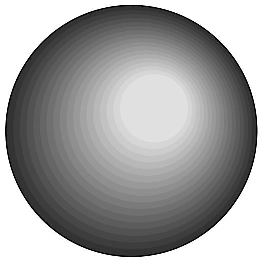
The ‘‘shell’’ variable contains the pathname to the current shell program.

See Also

csh(C), if(C), setfile(C), source(C)

ProLine

Networking



Name

arbitron, arbitron2 - Report weekly volume statistics on newsgroups

Syntax

arbitron [-a] [-v]
arbitron2 [-a] [-v] infile outfile

Description

Arbitron and *arbitron2* together compile statistics on the number of messages posted weekly to all newsgroups listed in the file $/etc/newsys.

Arbitron is a shell script which lumps all of the data from the $/sys/logs/*/newslog files (where * represents a day of the week) into one large file. *Arbitron* then calls the program *arbitron2*, which compiles the data and mails a summary to the root user.

Results are sent by email to the root account. They are designed to be launched by the *maint* or *maint2* script once per week. As an example, the line

```
if $time[0] = "Sun" then csh arbitron
```

could be placed near the end of *maint2*.

Normally, newsgroups are listed in the order in which they appear in $/etc/newsys. If the -a flag is used, they will be listed in alphabetical order. If the -v flag is used, they will be listed in order of volume (# of posts), with the most active groups listed last.

Note

Arbitron assumes that newslogs are rotated on a weekly basis (as is done with the standard *maint2* script). If a different method is used, the *arbitron* script must be modified.

Files

$/sys/logs/*/newslog - Newslogs for the past seven days

See Also

maint(C)

Author

Douglas Granzow (dig@pro-cynosure.clark.net)

Name

batch - Process mailboxes into newsgroups

Syntax

batch [-*x*]

Description

Batch is used by a site that serves as the central distribution point for a ProLine-based newsgroup. *Batch* takes messages that accumulate in mailbox files and converts them into newsgroup bundles stored in $spool/news. The bundles are further processed by *rnews* and *postnews* for distribution and local posting. To avoid additional processing, use the -*x* flag which causes *batch* to simply exit when it is done.

Batch finds newsgroup mailboxes by scanning through the $/etc/newsys file and matching them to mailboxes that exist in $/sys/mail. The mailbox names are defined in your sendmail aliases file ($/etc/aliases). Example:

```
pro-news-groups:          ~pro.news.groups
pro-comp-mac-networking: ~p.c.m.networkin
```

This sample from $/etc/aliases gives posting aliases for the groups pro.news.groups and pro.comp.mac.networking. Next to each alias is the name of the mailbox file. Mailboxes are designated as such by the use of the tilde character (see *aliases(NET)* for details). Mailbox names must agree with the naming conventions of the operating system. So they must start with a letter and be no more than 15 characters in length.

In addition, mailbox names must match the newsgroup aliases defined in the $/etc/newsys file. Newsys entries for the above aliases would be:

```
pro.comp.mac.networking=p.c.m.networkin
#D Macintosh networking discussion group
#L mac/networking
pro-sat!rnews, pro-finders!rnews

pro.news.groups
#D Discussion about ProLine newsgroups
#L pronews/groups
```

Notice the alias assigned to pro.comp.mac.networking -- p.c.m.networkin -- it matches the mailbox name given in $/etc/aliases. These must be identical for *batch* to find news mailboxes to process. If the formal newsgroup name is legal (15 characters or less and agrees with the operating system), like pro.news.groups, no newsys aliasing is necessary. See *rnews(NET)* for more details.

Batch should be called regularly via a crontab entry in order to process any mail that comes in. Unless the -*x* flag is used, *batch* invokes *unbatch* or *rnews* as appropriate to complete all processing and distribution of news.

Steps for Originating a Newsgroup

Understand that *batch* is used only to originate a ProLine newsgroup. You don't need to use *batch* if you only receive newsgroups from other sites (that's what *unbatch* is for).

1. Before starting a new group for distribution throughout the ProLine network (including USENET), you must first post a formal proposal for the group in pro.news.groups. Do not continue with these steps until your proposal is accepted and you're authorized to proceed.

2. Create an alias in your $/etc/aliases file so users can send mail to the group. Posting address are identical to the formal newsgroup name, except dashes replace the periods. Thus, to post a message to pro.news.groups, you send e-mail to pro-news-groups@pro-sol.cts.com.

3. The mail alias points to a mailbox file (as described above).

4. Add an entry for the newsgroup in your $/etc/newsys file.

5. Add the *batch* command to your $/etc/crontab file.

Files

$/etc/newsys - newsgroup description file,
$/etc/newslog - news processing log,
$spool/news/ - spool area for newsgroup bundles,
$/sys/mail/* - mailbox files.

See Also

aliases(NET), unbatch(NET), rnews(NET), postnews(NET)

Author

Morgan Davis (mdavis@mdg.cts.com)

Name

changesys - Automated newsgroup subscription service

Description

changesys is a news subscription manager that allows you to easily add and remove newsgroups sent to your system from your news source. Interaction with *changesys* is done completely through electronic mail.

Requests are sent to the host's *changesys* mailbox (e.g. *changesys@crash.cts.com*). Sites are assigned special passwords to verify requests to *changesys*. (NOTE: Only ProLine systems with *mdss* accounts on a news host are eligible to use the *changesys* service. Downstream ProLine sites should request newsgroups from their upstream ProLine neighbors.)

The message's subject field contains the name of the function to be executed (these functions are listed later). The body of the message contains lines identifying your system by name and password. The body can also include additional information if needed by the function specified in the subject field.

Here is the format of a *changesys* message:

```
To: changesys@my_news_host
Subject: function_name

system: my_site
passwd: my_password
```
(additional information follows)

Any additional information required by the command begins after the ''passwd:'' line.

The *changesys* functions are used to modify the news host's *sys* file: a file, like ProLine's *newsys*, that includes information about how newsgroups are distributed to neighboring connections. Unlike *newsys*, a UNIX hosts's *sys* file includes entries based on sites rather than newsgroups. Each entry in the *sys* file is specific to a particular site and lists all the newsgroups or newsgroup hierarchies that the site receives.

Services Provided

changesys recognizes the following functions:

help	Sends a list of *changesys* functions to you via mail.
getsys	Returns your current *sys* file entry via mail. This gives you the current listing of groups your site subscribes to.
getall	Returns your current *sys* file entry plus a list of the all newsgroups you receive based on your site's entry.

getgroups	Requests the current list of available newsgroups, sent to you via mail. This can be quite lengthy.
addgroups	Adds groups to your site's *sys* file entry, instructing the host to begin sending those groups to you. This command requires additional information in the body of the message (see below).
delgroups	Deletes groups from your site's *sys* file entry, instructing the host to cease distribution of those groups into your system. This command requires additional information in the body of the message (see below).
suspendsys	Effectively deletes your *sys* file entry, saving it for later retrieval with the ‘‘restoresys’’ command. Use this when you wish the host to temporarily cease all distribution of news to your system. While your *sys* entry is suspended, no news is stored for your site.
restoresys	Restores your previously suspended *sys* file entry.
delsys	Deletes your *sys* file entry completely, thus permanently halting news distribution to your site.

Subscription Lists

Some *changesys* functions require lists of newsgroup names. The subscription list format used on UNIX news systems follows the general syntax:

newsgroup-pattern,newsgroup-pattern,...,newsgroup-pattern

It is a comma-separated list, containing no spaces. A newsgroup pattern is a newsgroup name, or an initial portion of a newsgroup name specifying all newsgroups beginning with that pattern. A pattern may have ‘!’ as the first character, which causes newsgroups matching that pattern to be excluded.

Examples:

```
soc,rec,news.software.b     # all newsgroups in the "soc" and
                            # "rec" hierarchies, and the group
                            # "news.software.b"

rec.pets,!rec.pets.cats,fl  # all newsgroups in the "rec.pets"
                            # hierarchy except "rec.pets.cats".
                            # All groups in the "fl" hierarchy.
```

Adding and Deleting Groups

The ‘‘addgroups’’ and ‘‘delgroups’’ functions require additional information in the body of the message immediately following the ‘‘passwd:’’ line. The line begins with the function name, a colon, and a newsgroup subscription list as defined above.

Here’s an example using ‘‘addgroups’’:

```
To: changesys@my_news_host
Subject: addgroups

system: my_site
passwd: my_password
addgroups: ca.test,comp.sys.unix,sdnet
```

This adds two groups and one whole distribution (‘‘sdnet’’). To receive an entire branch of the news tree you need only to give the group’s base name (e.g. ‘‘sdnet’’). In the above example, everything in and under ‘‘sdnet’’ would be sent to your system. If you wanted to receive all of ‘‘sdnet’’, but not ‘‘sdnet.test’’, the ‘‘addgroups’’ line would be:

```
addgroups: ca.test,comp.sys.unix,sdnet,!sdnet.test
```

To cease reception of sdnet and its subgroups, your message would use the subject ‘‘delgroups’’, and the line following your password would be:

```
delgroups: sdnet,!sdnet.test
```

It is important to understand that *changesys* simply adds and deletes the items you specify -- it’s like text editing via remote control. It isn’t smart enough to know that if you removed only ‘‘sdnet’’ that you meant ‘‘sdnet.test’’, too. You have to include all group items you wish to remove.

You can add and delete groups in the same message by including both an ‘‘addgroups:’’ and a ‘‘delgroups:’’ line (the subject of the message can be either ‘‘addgroups’’ or ‘‘delgroups’’). Example:

```
To: changesys@my_news_host
Subject: delgroups

system: my_site
passwd: my_password
delgroups: alt.barney.dinosaur.die.die.die,\
alt.fan.alfred-e-neuman
addgroups: alt.bbs,!alt.bbs.internet,\
rec.arts.comics
```

Long group lists can be continued on multiple lines by ending them with a backslash character, as shown above.

Important

Use caution when changing your *sys* file entry. An erroneous entry may result in a dramatic increase in the news your site receives. Or it may result in your receiving no news at all. Using a ‘‘getall’’ command after adding or deleting groups is always a good idea.

The function names, including the ‘‘system:’’ and ‘‘passwd:’’ lines, must be spelled correctly. At least one space is needed between the colon and the parameter.

Note

The version of *changesys* described here is currently unique to CTSNET. Other versions do not have ‘‘addgroups’’, ‘‘delgroups’’ or any error checking for existence of groups.

Author

Original author (version 2.5), Christian Seyb; additions for CTSNET (version 3.0+), Bill Blue, CTS Network Services.

Name

domains - All about Internet domains

Description

In the Unix world there are things known as domains. Domains are basically specific machines that are knowledgable (mail-wise) of other machines in that domain. For example, the authority for the .com domain is the site called ucbvax. The domain that governs most of the ProLine network is called .cts, and is a second level domain or ‘‘park’’ (cts stands for Crash Time Sharing).

My site, crash, is the official registered authority for the .cts domain. So then, .cts.com really is crash, the authority that presides over that domain. Sites on any network that use what is known as Internet addressing, will know about the various registered domains -- at the very least .com, (ucbvax) and .cts.com (crash) as well.

The idea is that any site that is connected to a domain can be addressed via that domain. And any Internet site that knows about domains can basically get the mail through by looking at domain authorities. Domains are akin to ZIP codes we add to our postal addresses. With them, the Post Office can usually get mail to us even if the address is missing a city and state, and sometimes even your house number and street! Domain usage is similar, though an address has to, at the least, contain a user name, and the user’s home system name.

Mailing to bblue@crash.cts.com means ‘‘send mail to bblue at site crash in domain .cts in domain .com.’’ If a given Internet site does not recognize .cts.com, it will recognize .com and send the message along. The .com authority will know about .cts.com and pass it along. The crash!) and deliver the mail to bblue.

This also means that any site that crash knows about can be referenced simply in an Internet address. So bblue@pnet01.cts.com or bblue@pro-sol.cts.com are legitimate because crash knows about pnet01 and pro-sol, and crash is the authority of the .cts.com domain. Get it?

A domain-owning site, like crash, knows about all the sites under its influence because of the mapping system we use. For more details on maps and mapping, see the map(NET) manual.

See Also

map(NET), tutorial(NET)

Author

Bill Blue (bblue@crash.cts.com)

Name

intro - Introduction to networking

Description

This document describes the procedures for linking your ProLine system with ProLine and Unix sites. It does not describe how networking operates. It simply presents the steps required to configure your system to exchange network traffic.

0. Get Experienced and Organized

Learn to use the local mail and conferencing systems. Be sure you understand all facets of manipulating mail on your own system before you send mail to someone else's. Study all the manuals that discuss the networking commands and tools. You should at least know what there is to learn and where it can be found for reference later on.

Second, determine which site you should link up with first. Invariably, new sites all want to connect with pro-sol, the ProLine system run by the Morgan Davis Group. This is no longer a good idea. Pro-sol is currently connected to more systems than is good for it. With all these connections on a single-line system, it is nearly impossible to get past the busy signals. You would do well to seek connections with another site.

Third, once you choose a site, contact its administrator for permission. If all is well, exchange information with the administrator to establish a connection. Find out what the system's fastest baud rate is, when and how often it would be best to call, exchange identification codes (discussed later on), and make sure your spelling matches.

1. Create the MDSS Account

If you've already created the mdss account, skip to the next section.

To allow a remote ProLine system to call yours in order to exchange mail, you must create a special entry in your password file. This process has been automated by accessing a hidden menu from *plush*, the ProLine Users Shell and is documented in the Installation part of the ProLine manual. From the Main Menu in *plush*, type asterisk (*) to open the hidden Setup Menu. Press M to create the mdss account. When a ProLine site calls your system, it uses this ‘‘mdss’’ account.

2. Run the Mksite Script

The *mksite* script performs most of the steps required in getting connectivity set up. These include:

- o Creating an mdss directory

- o Adding an idfile entry (see *mdss* for details).

o Bringing up the L.sys file for you to edit (see *poll* for details).

Run the *mksite* script from the Networking Maintenance menu in *plush*.

3. Set Up the Scanner

When the previous items are completed, add an entry into your $/etc/crontab file using any text editor. The entry can contain the *scan* command as shown in this sample crontab file:

```
Day  ## Mon    HH MM Command
*    *  *      *  15 scan
*    *  *      03 20 csh maint -r
```

In this example, the *scan* command executes 15 minutes after the hour, every hour, though you can program it for any frequency you require. *Scan* searches the $/mdss site subdirectories for any mail to deliver. When done, it calls upon *poll* to make the connections, which in turn calls upon *mdss* to transfer the mail.

You could also enter a single *poll* command in your crontab file. This causes the site to be polled at the specified time whether or not there is anything to send. The command format would be ‘‘poll pro-site’’.

A better, though initially more complicated approach is to invoke a script containing scan, poll, or any other mail processing commands from your crontab file. Putting these commands into a script ensures that every step is processed. If they were individual cron entries, a task may be skipped if the system is busy with other work. Here’s an example of invoking a ‘‘mailtask’’ script:

```
Day  ## Month  HH MM Command
*    *  *      *  15 csh mailtask
```

4. Submit Your Map

After establishing a link with any site, you must submit an updated map entry file. See the map(NET) manual for details on creating, updating, and submitting your map.

5. Care and Feeding

You now know the step-by-step procedures for getting networked. Note, however, that there is more to it all than the steps listed above. ProLine sports a battery of utilities for maintaining and tracking network mail activity. You’ll want to familiarize yourself with these tools.

Setting up a link with additional sites is easy. You need only to run *mksite* the next time you wish to network with a new site.

Keep an eye on your log files. They can grow like a cancer and eat up all your disk space. It is a good idea to implement a ‘‘log file rotation’’ scheme as part of your automated daily maintenance activities.

Summary Checklist

0. Get Experienced and Organized
1. Create the MDSS Account
2. Add the site
3. Set Up the Scanner
4. Submit Your Map
5. Care and Feeding

Files

$/etc/L.sys - systems file for dialing and logging in,
$/etc/idfile - site identification file,
$/mdss/* - connecting site subdirectories,
$/sys/maps/* - storage of map entry files.

See Also

mksite(NET), rmsite(NET), maint(ADM), map(NET), mdss(NET), poll(NET), scan(NET)

Name

mdss - Mail Delivery SubSystem

Syntax

mdss [*options*] [*site*]

Description

Mdss interacts with a remote mail system to transfer mail files. It uses a binary protocol with error checking to exchange batches of files in both directions. *Mdss* is simply an agent for exchanging mail using a number of file transfer protocols.

When a remote site is called, *mdss* is invoked by *poll* passing along the polled site's name.

When a remote site originates the connection by logging in, *mdss* is invoked (from the interpreter field in the password file) without a *site* argument. The calling system must then identify itself. Its response must match an entry in the mdss idfile ($/etc/idfile). Idfile entries consist of a site name and password code, separated by a semi-colon (e.g., ‘‘pro-site;password’’).

Comments can be placed in the *idfile* following a pound-sign (#).

Mdss recognizes these options:

-d n	Delay for *n* seconds after receiving the command prompt from the slave. This is for MDSS implementations that cannot handle immediate responses from the master after it receives the ‘‘enquire’’ prompt and sends a response. Typically, such implementations would be those written for slower systems, MDSS interfaces built from shell scripts, etc.
-p or -u	Invokes standard XMODEM protocol with text filtering for use with P-Net hosts or older implementations that do not exchange total byte and job count information at the beginning of transfers.
-s	When everything is done, and only if files were received, invoke *sendmail(C)* before exiting.
-x flags	Transfer options. Specifies *flag* characters to be used by the file transfer protocol module invoked by *mdss*. One or more of the following characters can be given as the *flags* argument for XMODEM mode (default) transfers:

C	128 byte blocks with CRC
D	Doubles interpacket timeout delay tolerances.
K	1K blocks with CRC
L	4K blocks with CRC

X	Standard XMODEM, defeating ProDOS information packet handshaking. The ProDOS extension involves sending special packets of file information required by ProDOS systems that don't employ the Binary II file wrapping scheme.
B	Apple Binary II wrapping and unwrapping. The X flag should be included whenever Binary II mode is enabled to avoid sending two types of information packets.
T	Text filtering. Enables newline conversion and text filtering.

-y	Invokes YMODEM batch mode of XMODEM (you may still need to use the -x option to specify additional modes, such as Binary II, etc.). There is no advantage to using YMODEM instead of XMODEM, but the ability is included as a matter of completeness.
-z	Invokes ZMODEM batch mode.

Typical Options

Typical invocations for ProLine systems use the following options:

<none>	Omitting the -x, -y, and -z flags assumes standard 128-byte checksum XMODEM protocol with the ProDOS information packet handshaking (for backward compatibility with the original MDSS standard).
-x BLX	Enables Binary II, Large packets, and standard XMODEM modes. This is the preferred mode for use between two Apple II-based ProLine systems.
-z	Enables ZMODEM mode.
-x TLX	Enables text file filtering and 4K XMODEM modes, most useful when connecting with Unix systems that employ the UMDSS gateway software from the Morgan Davis Group.

An MDSS Session

After logging in successfully, the calling system assumes the roll of master, and commands the answering system to receive mail, if any exists. When there is no mail left to send, the master commands the answering site to send mail. At this point the master and slave reverse rolls.

During mail exchange, both sites scan the files received (or sent) to gather information for making log file entry. Two sample log file entries might include:

```
3/4-07:05:42 R 8968 pro-sat!wen
3/4-07:06:13 S 1421 pnet01!bblue
```

The entry begins with the date and time (dd/mm-hh:mm:ss). The letter S indicates mail sent, while R indicates mail received. Following, is the number of bytes delivered, and the addressee's name. Mail received is placed into a spool directory for local delivery by *sendmail(C)*.

A work summary is displayed during the session showing jobs counts, byte counts, the effective characters-per-second rating, and an efficiency percentage. The efficiency rating is determined by comparing the number of characters sent per second against the connection speed. For example, 960 cps on a 9600 baud connection is 100% efficient. On modems with data compression, it is not uncommon to see better than 100% efficiency.

Once all mail has been exchanged, the master gives the hangup command, and both sides disconnect. If any mail was received, and the *-s* option is included in the invocation, *mdss* runs *sendmail* before exiting.

Note

Active systems should keep an eye on the fast-growing mdss log file. It should be pruned occasionally, or archived offline, in order to save disk space.

Files

$/etc/idfile - mdss password file,
$spool/logs/mdsslog - mdss log file,
$/mdss/* - remote site directories,
$spool/mail - received mail spool area.

See Also

intro(NET), poll(NET), scan(NET), sendmail(C)

Name

mdssclean - Clean network site directories

Syntax

mdssclean [*options*] [*site*...]

Description

Mdssclean cleans the $/mdss directories by deleting old rnews batches and returning old mail. It also removes any ‘‘Finder.Data’’ files found, and compacts empty site directories to reclaim disk space.

When invoked without any *site* arguments, *mdssclean* automatically cleans all directories found in $/mdss. Otherwise, *mdssclean* only processes the directories specified by *site* arguments.

Mdssclean obtains user-defined default settings from its resource file (described later), but these can be overridden by supplying command line options.

Options are:

-d	Enables deleting. Without this option, *mdssclean* goes through the motions, reporting file ages, etc., but does not bounce and delete messages, nor compact directories.
-r	Operate only on rnews batches (skips mail).
-a age	Sets the age limit on files in days. Files the same age or older than this setting are eligible for cleaning. (Default is 10 days)
-i	Include invisible directories when *mdssclean* is invoked without any *site* arguments.
-l lines	Sets the number of body *lines* to return in bounced messages. Does not apply to messages sent with *rcp*; such files only have the header and rcp information stub returned. A value of -1 returns the entire message. A value of 0 returns only the header. (Default is -1).

Resource File

The *mdssclean* resource file includes three items. The first item is the number of days to assume as the *age* for deleting files. The second item is the maximum number of body lines to return. The third item is a list, beginning with a count, of names to ignore when bouncing a message. For example, if old mail is from ‘‘uucp’’, it should not be bounced back to ‘‘uucp’’ since it isn’t a real person.

Sample resource file:

```
10
-1
3
```

```
          mdss
          uucp
          postmaster
```

Mdssclean's internal defaults are the same as the above example.

See Also

maint(ADM), mdss(NET)

Files

$/mdss/* - site directories,
$/etc/rsrc/mdssclean.rsrc - resource file.

Author

Morgan Davis. Original author Daniel Davidson.

Name

mknull - Make remote letters addressed to null

Syntax

mknull [-a] [-f] [*site*...]

Description

Mknull allows the system administrator to make ‘‘null’’ files in the mdss directories of directly connected sites. The presence or absence of a file allows *scan* to determine whether to poll a connecting site.

If the -a option is provided, null files are created for all sites directly serviced by the system. If the -f option is provided, null files are created only for sites which already have traffic pending.

The Null File

A ‘‘null’’ file contains a simple mail header, addressed to ‘‘null’’ at the remote site. Null files are sent from the system administrator to the null alias at connecting sites. They are handled by the *sendmail* program at connecting sites just like letters.

Here is a sample null file produced by *mknull* at a site (pro-example) that has a direct connection with pro-sol:

```
From pro-example!root Sat Jun 22 11:23:37 1991
Date: Sat, 22 Jun 91 11:23:37 EDT
Ppath: pro-sol!null
From: root@pro-example.cts.com (System Administrator)
To: null@pro-sol
Subject: null
X-Mailer: Mknull (1.1 8dec91)
```

This null file is named $/mdss/pro.sol/a0. If it were directed to pro-carolina, it would be named $/mdss/pro.carolina/a0, and the lines of the null file which contain ‘‘pro-sol’’ would instead contain ‘‘pro-carolina’’.

Scanning for Null Files

Null files can be used by *scan* to determine whether or not to poll another site. *Scan* causes *poll* to call all sites that have null files in their $/mdss directories if it is called in the following manner:

```
scan -il -f a0
```

Assuming the null file, shown above, exists, and the system polls pro-sol, the null file would be sent to pro-sol!null, and it would no longer exist in the $/mdss/pro.sol/ directory. Thus, the next time scan runs, pro-sol would not be polled. In fact, pro-sol would not be polled again until another file is created. Thus, null files can be used as one method to LIMIT how often to poll another site.

Resource File

If no *site* arguments are given, mknull reads the resource file $/etc/rsrc/mknull.rsrc. Each line of this resource file contains the name of a connecting site for which a null file should be made. Example:

```
pro-carolina
pro-sol
```

This tells *mknull* to make the files $/mdss/pro.carolina/a0, and $/mdss/pro.sol/a0.

Note

If files already exist for a given site, *mknull* renames an existing queued file to ‘‘a0’’ instead of producing a null file. If only rnews batch files are queued, *mknull* produces a separate null file.

Notify the system administrator of each connecting site that you intend to send null files to their site. They must have an alias which directs all null files to a mail file that is regularly deleted, preferably by a maintenance script.

Files

$/mdss/*/a0 - null files produced by mknull,
$/etc/rsrc/mknull.rsrc - resource file.

See Also

aliases(NET), scan(NET), sendmail(C), rmnull(NET)

Author

Dean Fick (dean@pro-electric)

Name

mksig - Make signatures

Syntax

mksig [*options*] [*user*]

Description

Mksig generates user signature files based on the signature template file $/etc/default/signature. To generate or update your own signature file, enter *mksig* without any arguments.

The super user can generate or update a signature for a user by including a *user* name argument.

Other *options* available to the administrator (as long as there are no more than 1000 accounts):

-a Generates signature files for all users.

-e Updates existing signature files for all users.

The Signature File

Users can have their own *signature* files created only if the default signature file exists. When a user sends a message to an offsite address (into the network) a signature file is appended to the end of the message. This file gives information on the user's address and position on the network.

The default signature file can contain any text, including special keywords (below). When *mksig* creates the signature file, it copies the default signature exactly, substituting the user's information when any keywords are encountered. The keywords are:

<NAME>	Full name (first and last)
<LOGIN>	Login name
<HOST>	Local host name
<DOMAIN>	Domain for the local host

It is important to keep signatures as short and as simple as possible. Long, garish, space-wasting signatures are indicative of unprofessional, inexperienced network users.

Notes

Mksig runs slower if the signature template contains the <NAME> keyword, causing it to search $/etc/passwd for the user's real name.

Mksig refuses to run if the signature template is longer than 10 lines.

Files

$/etc/default/signature - signature template.

Author

Dean Fick

Name

mksite, rmsite - Make or remove network connections

Syntax

mksite
rmsite [*site*]

Description

A set of shell scripts, *mksite* walks you through the process of setting up networking information for a site while *rmsite* removes that information when network connectivity is no longer needed.

Mksite does most of the steps required in getting connectivity set up. It does not write the L.sys entry, but can bring up the L.sys file in your editor.

Rmsite, likewise, does everything except remove the site’s L.sys entry.

You also need to edit and resubmit your updated site map.

See Also

intro(NET)

Author

Morgan Davis (mdavis@pro-sol.cts.com)

Name

ned - Newsys file editor

Syntax

ned

Description

Ned is a convenient editor for managing the system's master news distribution file, *$/etc/newsys*. This program is still under construction. Right now, only the <U>pdate command is useful.

Files

$/etc/newsys - the newsys file.

See Also

batch(NET), postnews(NET), rnews(NET), unbatch(NET)

Author

Morgan Davis (mdavis@mdg.cts.com)

Name

netauth - Authorize network mail access

Syntax

netauth [*options*] [*user* ...]

Description

Netauth is an administrator's tool that manages network mail access. Network access is determined by the existence of a special file in each user's $/adm directory, called ‘‘mailstop’’. If mailstop exists, the user is denied network mail sending privileges. This tool simplifies and automates many tasks related to managing network mail access.

The administrator uses *netauth* to grant or deny network mail access for one or more local accounts. It can be used to automatically grant network access after a probationary period. Furthermore, *netauth* can report network access for any or all accounts.

Netauth takes the following options:

*	Reports net access status for all users.
?	All about *netauth*.
user	Reports net access status for the given user.
-a user	Allows net access for the given user.
-r user	Removes net access for the given user.
-t user	Same as -r, except allows auto-aging.
-d [n]	Grants network access by deleting all mailstop files older than 30 days (the default), or older than n days if a numeric argument is given.

Since *netauth* is most often used from a cron process, it generates and mails a log of its activity to the root account.

See Also

adduser(ADM), mail(C)

Author

Daniel Davidson (danield@pro-grouch).

Name

path - Show the mail route to a site

Syntax

path *site*

Description

Path shows the path through which mail is delivered to a specific *site*. For example:

```
path pro-electric
```

might produce a display like this:

```
pro-electric: pro-sol!pro-carolina!pro-electric!*
```

This means that a letter to a user at pro-electric would first be sent through pro-sol, through pro-carolina, and finally to the addressee at pro-electric.

Files

$/etc/paths - routing database for this site,
$/etc/rsrc/sendmail.rsrc - path to smart host.

See Also

aliases(NET), sendmail(C)

Author

Dean Fick (dean@pro-electric.cts.com)

Name

poll - Poll a site

Syntax

poll [-i] [-p *progress*] *site*

Description

Poll dials *site* to exchange mail, news, and files. It uses an extensive set of macro commands for dialing, connecting, and logging in. Information for connecting to a site is stored in $/etc/L.sys.

The L.sys File

The L.sys file consists of entries, one or more per site. The first line in an entry begins in the leftmost column, and starts with the name of a site to poll. Succeeding lines for the entry begin indented by a tab or spaces.

Entry lines contain space-delimited arguments, so any arguments requiring spaces should be enclosed by quotation marks (").

A sample entry might look like this:

```
pro-sample Any 2400 555-1212 local
        \p1
        \[login:] mdss
        \[word:] whatever
        \[id:] pro-test;password
        \!"mdss -x BLX pro-sample"
```

This entry consists of a main entry line and succeeding lines (indented) which continue the information for the entry. The first line begins with four fields of information necessary to begin the poll attempt.

Comments can be included following a pound-sign (#).

L.sys Fields

The first four fields are:

pro-sample	The name of the remote site.
Any	A valid time specifier. This includes one or more of the following keywords or time ranges:

Any	Site can be polled at any time.
Dead	Site is never called unless *poll* is forced to ignore time restriction information by including the -i command line flag.

Never	Site is never called, even if -i is given.
Sun	(and Mon, Tue, Wed, Thu, Fri, Sat). The site can be called only during the days specified.
End	Site can be called during the weekend (Sat and Sun).
Day	Site can be called during a weekday (Mon through Fri).
xxxx-yyyy	Site can be called during the specified time range. Times are given in 24 hour format and must contain four digits (no colons). For example, 0000-0630 means the site can be polled between midnight and 6:30am.

Multiple time qualifiers can be listed, each separated by commas (no spaces). For example, ‘‘Tue,Fri=1800-2000,End’’ means that a site can be polled any time during Tuesday, only between 6pm and 8pm on Friday, and any time during the weekend.

A single time range cannot cross midnight, but a pair of ranges can be given to do this (e.g. ‘‘2330-2359,0000-0330’’).

2400 The fastest common speed on both systems.

555-1212 The phone number of the remote site. An alternate number can be specified for multi-port sites with differing phone numbers. This is done by separating the two numbers with the pipe (|) character (e.g., ‘‘555-1234|555-5678’’).

A phone number can begin with ‘‘AT’’ to signify a custom dialing command (e.g., AT&M4DT555-1212). This allows a command to be sent to the modem before dialing. Note: If you have an MNP-style modem that uses backslash (\) commands, include two backslashes where normally you would use only one (e.g., AT\\N0DT555-1212).

If the site does not answer, *poll* dials a second time, using the alternate number if given.

local An arbitrary identifier classifying the type of connection. The name you pick is totally up to you -- it is meaningful to you only. Typical examples might be ‘‘local’’ (a local call), ‘‘toll’’ (long distance), etc. The identifier can also be the name of a long distance carrier server such as ‘‘ATT’’, ‘‘Sprint’’, ‘‘MCI’’, ‘‘PCPursuit’’, etc. The identifier cannot include spaces or any characters illegal to the operating system.

Macro Commands

After the fourth field, the macro commands begin. Macro commands start with a backslash and one character. The macro commands are:

\[Pattern match. *Poll* suspends macro execution, watching all the incoming characters until it matches the pattern given between the left and right square brackets. Example: \[login:]

By default, \[waits 45 seconds. You can adjust the duration by following the last bracket with a value in seconds. Example: \[text]90.

The \[command also supports a UUCP-style ‘‘expect-send-expect’’ feature. This embodies two pattern matches and a sending command in one. For example:

```
\[hello]-hi-[hello]
```

This tells *poll* to wait for ‘‘hello’’ first. If it receives it within the duration specified, *poll* skips the rest of the macro and goes onto the next one. If ‘‘hello’’ is not found in time, ‘‘hi’’ is sent (with a carriage return on the end), and *poll* once again waits for ‘‘hello’’, the second pattern.

Failure to match a pattern cancels the *poll* attempt.

\d One-second delay. If an argument is used, such as with *\d2* it denotes the number of seconds to delay (in this case, two seconds).

\b Send a modem break tone.

\c Clear the input buffer. This is used to flush any buffered input from the host system. It would be useful in a situation where buffered input could trigger a handshake to occur at the wrong time.

\p Set progress reporting (unless overridden by the -p option on the command line). By default, *poll* displays a ‘‘live’’ login session while connecting with a site. By turning on progress reporting with *\p1*, characters sent from the polled site are not displayed. Instead, *poll* reports progress messages that can be very useful in debugging a connection attempt. Using *\p* or *\p0* turns off progress reports.

The -p command line option allows you to override any \p commands. For example, including ‘‘-p 0’’ or ‘‘-p 1’’ in the command line causes *poll* to set the progress mode to on (1) or off (0) and ignore any \p macros.

\! Launch application. This invokes a network protocol to use after all the handshaking has been done. If the command line requires arguments, the entire command line must be enclosed between quotation marks, as in the example given earlier.

Mdss is used for ProLine systems and requires at least one argument: the name of the site being polled. See *mdss* for details on the options it accepts.

Any argument that does not start with a backslash (\) and a command character is sent to the remote site. A newline is automatically sent, unless the argument ends with a semi-colon (;).

Important Notes

All arguments for an L.sys entry are delimited by space characters. Thus, if you need to match a pattern that contains one or more spaces, enclose the argument in quotation marks:

```
    \["net login:"]
```

Any control characters needed in an argument can be represented by using the caret (^), as in ^C for Control-C. The backslash (\) can be used to escape special characters like the quote ("), caret (^), and backslash (\).

The L.sys file can contain more than one entry for a single site. This might be desirable for using a different carrier or long distance service depending on the time or day of week. *Poll* scans the entire L.sys file for matching site entries until it finds one that has a valid poll time and successfully connects.

Files

$spool/logs/mdsslog - a report of network mail activity,
$/etc/L.sys - remote site macro file,
$/mdss/* - directory containing site subdirectories.

See Also

mdss(NET), scan(NET), sendmail(C)

Name

postnews - Post newsgroups to local news areas

Syntax

postnews

Description

Postnews is invoked by *rnews* when there are newsgroup bundles queued in the $spool/news area. Bundles are posted into the local news areas (conference system topics). Since *rnews* calls *postnews* when there is work to do, there is no reason to invoke *postnews* from the shell or from a crontab entry.

Postnews delivers newsgroup bundles based on information in the $/etc/newsys file. (See the *rnews* manual for details on the newsys file format).

Postnews Resource File

The resource file for *postnews*, $/etc/rsrc/postnews.rsrc, contains a list of newsgroup field names to be retained in messages posted in the Conference System. A typical list of fields looks like this:

```
Newsgroups
Summary
Keywords
Message-ID
Followup-To
References
Sender
Reply-To
Distribution
Organization
Lines
```

To instruct *postnews* to leave out certain fields, edit the resource file and place a pound-sign (#) before each field to be ignored. The Date, From, and Subject fields are always included in CS messages, and therefore do not need to be in this list.

Files

$/etc/newsys - newsgroup distribution file,
$/etc/rsrc/postnews.rsrc - resource file containing field list,
$spool/news/* - newsgroup bundles.

See Also

cs(C), cs.maint(C), ned(NET), rnews(NET), unbatch(NET)

Name

rcp - Remote file copy

Syntax

rcp [-abq] [-f from] [-s *subject*] *file user@site*...

Description

Use *rcp* to send text or binary files (of any type and length) to a user on a remote system. The syntax is similar to the standard *cp* command which copies a file on the local system, only you specify a mail address in the user@site format as the target. Multiple *users* can be specified, but multiple *files* cannot.

Options:

-a	By including the -a option, *rcp* sends an ASCII text file as one letter to the recipient. The text file appears as one message in the user's mailbox.
-b	Binary mode (the default). Binary mode works only between point-to-point ProLine sites. Intermediate non-ProLine sites will corrupt the file. The recipient receives an accompanying letter, generated by *rcp*, in his mailbox, reporting the status of the remote copy transfer. The binary portion is saved into his user directory with the same name as the original file, and all file attributes are preserved.
-f from	Allows the super user to specify the original sender.
-q	Quiet mode.
-s	*Rcp* uses a the name of the file being sent as the subject. If the -s option is given, the argument that follows becomes the *subject*.

Note

The system administrator can deny a user reception of *rcp* files by creating a file called ‘‘rcpstop’’ in the user's $/adm area. This is desirable with public accounts.

Directory files cannot be sent.

Warning

The user should exercise caution in sending binary files without knowing the path the file will take through the network. Raw binary (8-bit) files are supported only by ProLine mail systems. Should such files pass through a non-ProLine system that strips high bits and control codes, the recipient will receive a corrupted file. To ensure success in all cases, binary files should first be archived and then encoded into a text (7-bit) format before being sent with *rcp*. The *uuencode* utility provides such a feature.

See Also

par(C), path(NET), uuencode(C)

Name

rmnull - Remove null files

Syntax

rmnull [-a] [*site*...]

Description

Rmnull provides the system administrator with an efficient way to remove null files that have been created with *mknull*. For specific information about null files, see *mknull*.

When invoked with the -a argument, *rmnull* searches through all connecting site subdirectories in the $/mdss subdirectory, looking for files named ‘‘a0’’. When such a file is found, it examines the mail header of the file to determine whether or not it was created by the *mknull* command. If so, *rmnull* deletes it. If an actual letter file was renamed to a0 by *mknull, rmnull* renames the file to its original name.

rmnull may also be called to work on specific connecting sites, rather than all of them, by giving one or more *site* arguments. Example:

```
rmnull pro-sol pro-carolina
```

rmnull looks through the $/mdss/pro.sol and $/mdss/pro.carolina subdirectories only, removing any null files found there.

Files

$/mdss/*/a0 - null files removed or renamed by *rmnull*.

See Also

mknull(NET), scan(NET)

Author

Dean Fick (dean@pro-electric.cts.com)

Name

rnews - Distribute newsgroup bundles

Synopsis

rnews

Description

Rnews distributes newsgroup bundles found in $spool/news as specified by the $/etc/newsys file. Bundles are sent to offsite addresses as whole batches of articles in one file, or the bundles may be broken into individual articles for standard mail distribution, letter by letter (this is not as efficient as forwarding whole batches).

Rnews is invoked by *unbatch*. It isn't useful to invoke *rnews* from the shell or via a crontab entry.

Sample Newsys File

The news unbatching, distribution, and posting system uses $/etc/newsys to determine how and where newsgroup bundles are distributed. A typical newsys file looks like this:

```
comp.sys.apple2
#D Discussion about Apple II micros
#L apple/net 150
pro-simasd!rnews, ~pro-tcc!apple-local

comp.sys.mac.programmer=c.s.mac.prog
#D Macintosh programmers' group
#L macintosh/coding 150

comp.dcom.telecom=c.dcom.telecom
#D Telecommunications
#L telecom/general
pro-simasd!rnews
```

Format of Newsys

The newsys file consists of one more group entries, separated by a blank line. Each entry includes a number of fields.

The first field in an entry gives the name of a newsgroup. Group names must be legal file names (15 characters or less and contains no illegal characters). If the group's name is not a legal file name, it can be aliased to a legal one with the following syntax:

real-long-group-name=legal.name

For example, comp.sys.mac.programmer could be aliased to c.s.mac.prog, rec.arts.startrek could be r.a.startrek, and news.announce.newuser could be n.a.newuser.

It is recommended that after adding or editing groups in the newsys file that *ned* be run to update the newsys file. *Ned*'s update feature makes sure that groups that need aliases are given legal, descriptive names.

The fields which follow consist of either target addresses or special information for that entry.

Example Newsys Entry

Study this sample entry:

```
comp.sys.apple2
#D Discussion about Apple II micros
#R incoming from cts.com
#L apple/net 150
pro-simasd!rnews, ~pro-tcc!apple-local
<pro-frisbee!rnews>, pro-offline!rnews
```

This entry for comp.sys.apple2 begins with three fields of special information:

#D Description. This is a short description that summarizes the group's content.

#R Remark. The #R field is a general note to yourself, such as a note indicating which site feeds you with the newsgroup. Extra #R fields can be included for more comments.

#L Local Info. The #L field gives local Conference System posting information. It contains the name of a conference/topic that gets copies of the articles for that newsgroup. An optional numeric argument can also be included to denote the maximum number of articles to remain active in that conference. This information is used by *postnews*.

The letter following the pound-sign *must* be CAPITALIZED.

Finally, in the example, the last two lines contain a list of addresses for sites that subscribe to the newsgroup. More than one address can be included on a line. Each address must be delimited by a comma, followed by a space. Use as many lines as necessary, but don't leave any blank lines between addresses.

A site can be removed from distribution by placing a left-angle bracket (<) in front of the address -- an enclosing right-angle bracket (>) is optional. This stops *rnews* from feeding a site while leaving it in the list.

If an address starts with a tilde (~), the articles are sent one at a time in standard e-mail letter format. If the address does not begin with a tilde (~), the articles are sent in batches (multiple articles per file). Batches are always addressed to a site's ‘‘rnews’’ account.

Log File Entries

The three programs, *unbatch*, *rnews*, and *postnews*, put entries into the log file *$spool/logs/newslog*. *Unbatch* reports newsgroups it unbatched and the number of articles taken from each. *Rnews* reports which addresses where newsgroup batches were sent. And, *postnews* reports how many articles in each newsgroup were posted to a conference/topic.

If a newsys entry does not send copies of a group to any addresses, and does not include a #L field (local Conference System posting information), then the newsgroup bundle will accumulate articles. An appropriate warning message is placed in the newslog file.

Files

$/etc/newsys - newsgroup distribution file,
$spool/logs/newslog - news processing log file,
$spool/news/* - newsgroup bundles.

See Also

ned(NET), postnews(NET), unbatch(NET)

Name

scan - Scan network mailboxes

Syntax

scan [-ailx] [-f *file*] [-o *flags*] [-p *app*] [*site*...]

Description

With no arguments given, *scan* looks in all directories in $/mdss finding any that contain files. Sites with outbound mail are polled in order to deliver the mail. Each site with mail is polled in succession until all sites have been processed.

If one or more *site* names are given, only those sites are scanned.

By using the optional arguments, *scan* can do more than just invoke *poll* when files are found. The options are:

-a	Force *scan* to think that all site directories contain files.
-i	Scan invisible directories, too. Normally, only visible directories are scanned.
-f file	Match only the directories that contain the named file.
-l	(L) List sites with work pending, including job counts.
-o flags	Option flags to pass on to *poll* (or the application specified by the -p option). The flags should be enclosed in quotation marks so that *scan* won't attempt to process them.
-p app	Sets the application to invoke on sites with work pending. (default is *poll*).
-x	Scan, but do not invoke *poll* (or the -p program).

Examples

```
scan -a -o "-i"
```

This example calls ‘‘poll -i’’ for all visible site subdirectories in the $/mdss directory, thus forcing the system to dial all sites regardless of time restrictions.

```
scan -lx pnet01
```

Scans $/mdss/pnet01, reporting any jobs pending. It will not execute *poll*.

```
scan -il -f a0 -p uustat
```

Scans all directories (visible or not) for those containing the file ‘‘a0’’. A list of those found is displayed before the uustat program is called to service them.

Notes

Scan is best used when called from a timed *cron* process.

Scan builds a list of sites that meet the criteria based on the options provided. This list is written out to shell script file, e.g.:

```
poll -i pro-gort
poll -i pro-klatu
poll -i pro-barata
poll -i pro-nikto
```

After building the script, *scan* executes the C-Shell to process the script.

Files

$/mdss/* - directories for networked mail sites,
$tmpdir/scan.x - temporary shell script.

See Also

cron(ADM), mknull(NET), poll(NET)

Name

tutorial - Networked e-mail tutorial

Description

This system is one node of the ProLine network. ProLine is the name of the software that turns an Apple II computer into this BBS. Through the ProLine network, users of this system can exchange information with worldwide networks, known collectively as the ‘‘Internet’’. ProLine users can also communicate with people on commercial services such, as America Online, AppleLink, BIX, CompuServe, Delphi, MCI Mail, and The Well which are also connected to the Internet. As a preface, there are some critical things that you should know about the ProLine network:

o Every ProLine system is an Apple II-series computer with a single phone line and usually a single hard drive. By today’s standards, these computers are tiny and slow, and it is indeed amazing that they can operate a system as sophisticated as ProLine and be part of a planetary network.

o When you send or receive a network message, you are costing every system along the way time, resources, and money. When you send or receive network data, you cost your system administrator money. You also cost the other ProLine sysops money. Please keep that in mind. This is not to discourage reasonable use of the network, but remember:

o Any abuse of the network will result in denial of network privileges.

o Finally, be conservative. If you don’t absolutely KNOW an address, don’t send the letter. Don’t ask for huge files, or send them. Be reasonable. If you’re in doubt, ask your system administrator.

Networking

Mail networking is simple in concept. You want to send a message to someone who has an account on a system other than this one. Although the idea is simple, the network’s operation is quite complex. Fortunately, most of this complexity is handled by the computers that make up the network. Nevertheless, there are a few things that you should know about ProLine lest this tutorial confuse you all the more. These include:

o A working knowledge of *mail* (where you can read or send mail), editors (*edit*, *vedit*, or *ed*, the ProLine text editors), and *cs* (Conference System). If you have already read mail, sent mail to your system administrator or to another user of this system, and if you have posted and read messages on the Conference System, then you have such knowledge.

o An understanding that you own a user area on the system’s hard drive for as long as you have an account on this system. Your user area is a subdirectory (or folder) inside the system directory called $/usr. It has the same name as your login name. If your login name were jsbach, for example, your user area would be $/usr/jsbach. Within your user directory are all your files, including an important file called *signature*.

- o Optionally, the use of some utilities on this system that aid in mail routing and file management. These include add, cat, cp, find, grep, rcp, rm, safecom, setfile, uuencode, uudecode, and a few others. These commands are accessed from ProLine's command-line, the C-Shell. This is not to say that you must be totally conversant in the use of all of these utilities, but they can make things a lot easier. Review the manuals on these commands to gain a basic understanding of the functions they perform.

Don't let this all overwhelm you. You can learn networking bit by bit, and soon you'll be a network expert.

Network Addresses

An address in the network is nothing more than a computer network path that you send a message through to reach a user on a distant computer system. The concept of addresses is rather straightforward, but the implementation of them can stagger the mind. ProLine, however, alleviates much of the complexity that would otherwise confound most users.

Without ProLine's smart routing features, you would have to know the complete address path that the message would follow to reach its destination. Consider the postal system as we know it today. Without relatively short addresses and ZIP codes, we would have to include complete instructions for letters carriers to deliver a piece of mail. ‘‘Drive from the post office to Elm Street. Take a right. Go up one block. Turn left at the signal. Fourth house on the right; the one with the pink Ford Escort in the driveway.’’ Now, consider out of town delivery instructions! What happens when whole cities (sites) move and change their routing paths (connections)?

To apply this example to electronic mail delivery, you would have to tell each computer in the network exactly through which systems your message should be relayed. For example, to send to a user named gbush at pro-applepi, a ProLine site in Washington, DC, from pro-sol, a site in San Diego, CA, you would have to know this address:

```
pro-pac!pro-freedom!pro-novapple!pro-applepi!gbush
```

As you can see, an exclamation point (!) separates each system in the path, just as a certain character separates directory names in pathnames under your computer's operating system. Needless to say, this isn't very convenient, and that lengthy address was needed to send a letter to a fellow ProLine site! Why can't the computers keep track of how to send the mail? The answer is, they can.

Domain (or Internet) Addresses

Fortunately, sites that make up the world computer network have devised simpler forms of addressing. These abbreviated addresses are known as ‘‘domain’’ or Internet addresses. A domain is simply a large computer that knows the explicit paths for a group (domain) of addresses. If you wanted to send mail to someone on BITNET, the academic network, you would simply send it to UserName@SiteName.BITNET.

To send to a user named JSMITH who has an account on the system CLARGRAD, for example, you would send to JSMITH@CLARGRAD.BITNET.

To send to jscully who has an account on a system at Apple Computer, you would just send to jscully@apple.com.

In addition, this system knows the path to every other ProLine system. You don't even need the domain identifier (the part of the address following the ‘‘.’’). To send mail to jbush at pro-applepi, you would address your message to:

```
gbush@pro-applepi
```

It's just that simple.

You will most likely learn someone's address by their telling you in person, or by reading it in their signature on a network message.

Signatures

Signatures are succinct addenda to messages you send to people on the network. They contain your address information, allowing persons wishing to respond or correspond with you to find the easiest mail path to you. The signature file is kept in your user area with the filename of *signature*. When you signed on to ProLine to create your account, a signature file was prepared for you.

Your signature automatically attaches itself to all correspondence that you send to the net or to any person on the net. So, the utmost consideration must go into choosing what you want to have as a signature if you desire to customize it. It should:

1. Provide a reasonable return address/path to you so that all persons reading your message will be able to send you mail.

2. *Not* contain any offensive text or diagrams in it that would detract from normal standards of conversation. This includes *any* vulgarity, and any vulgar text art.

3. *Not* be large. Four lines should be the most that you need for any signature file.

4. *Not* take up bandwidth. What this means is that if you add to your signature, the additional information should be worth the increased cost to distribute it through the network.

Summary

As you can see the interaction available in a network of this sort can be quite amazing! You can have correspondence going with a number of people around the world. But please do not abuse the system. The limitations of a small computer such as this one precludes us from having a really large system, and therefore the only way that you can cut the drive use down is to use it effectively. Please do not subscribe to any mail-feeds without prior approval from the system administrator. Anyone abusing the network privilege is likely to lose network access.

Files

$home/signature - contains your net address info

See Also

domains(NET), mail(C), rcp(NET)

Name

umdss - Unix Mail Delivery SubSystem

Syntax

```
umdss [ site ]
umdss -q
umdss -v
```

Description

Command line option -q queries umdss to report information on traffic pending for each site, including live connections. The -v option displays the umdss version and copyright information. With no arguments, umdss prepares for a remote site to log in and begin a session. With a site argument, umdss begins a session for a site that has just been polled (originate mode) and is connected. (Polling is not part of the umdss software).

Introduction

This document describes the umdss program that runs on UNIX computers, providing a compatible mail exchange interface with ProLine systems. It is organized into the following sections:

o About UMDSS
o How it Works
o Account Installation
o Installing UMDSS
o File Permissions and Directories
o ProLine Site Installation
o Transporting Mail
o Transporting Mail with Shell Scripts
o Transporting News
o For the ProLine Administrator
o Logfile
o Job and Session Reports

About UMDSS

Umdss is a simple mail exchange program. It implements the Mail Delivery Subsystem (MDSS) protocol designed in 1985 by Morgan Davis and Bill Blue for the purpose of simplifying the exchange of mail and news with microcomputers, like the Apple II.

Here are some of umdss’s features:

o Mimics UUCP in many ways.
o Maintains site lock (LCK) files.
o Updates session ‘‘.Status’’ files.
o Includes a uustat-like function (the -q option).
o Integrates with the local mail system (works best with smail 3.x).
o Maintains a detailed log file.

o Includes a large packet protocol (mdx) for 90-95% transfer efficiency.
o Supports non-batched external protocols as well (e.g. sx/rx).
o Compatible with high-speed modems and ports.

Note that umdss is simply a mail tranportation utility. It does not possess the ability to dial a remote host, connect, and login. A separate utility could accomplish this, then invoke umdss (with the name of the site being called as an argument).

How It Works

A ProLine system dials into the UNIX machine to login as ‘‘umdss’’. This runs the umdss program (with no arguments). The ProLine system is initially the master, commanding the UNIX machine (the slave) to receive mail (if there is any). When all mail is sent, the two machines reverse roles; the UNIX side becomes master, and ProLine, the slave. Again, the master commands the slave to receive mail (if there is any).

When a letter is sent, umdss calls upon an external file transfer program to transmit the file. Umdss deletes each file successfully sent, and places an entry in the umdsslog file.

Likewise, when a letter is received, umdss runs a transfer utility to receive the file, then mails it through rmail. When rmail is done, umdss logs the reception of the letter.

Among other things, log entries show the site, date, time, size and recipient of each letter. (See the ‘‘Logfile’’ section for more details).

Finally, when all mail is exchanged, both machines disconnect.

Account Installation

The first step is to create an account with the following attributes:

```
Login:          umdss
Password:       whatever
Home:           /usr/local/lib/umdss
Shell:          /usr/local/lib/umdss/umdss
Aging:          Disabled -- the account should never expire.
```

While you’re at it, set up a mail alias so that any incoming messages addressed to ‘‘umdss’’ (i.e. bounces from the net) are sent to a legitimate administrative account.

You should include umdss in the same group as ‘‘uucp’’.

Installing UMDSS

The umdss package, normally distrbuted as a compressed tar file, includes C source code, utilities, and support files for running umdss on a UNIX computer.

Follow these steps to install the UMDSS package (it is assumed that you have sufficient access to create directories and accounts on the system):

1. Uncompress umdss.tar.Z (e.g. uncompress umdss.tar.Z)

2. Change directories to /usr/local/lib (e.g. cd /usr/local/lib). (The recommended location is in the /usr/local/lib hierarchy. This location is used throughout this document).

3. Unarchive the tar file (e.g. tar xvf umdss.tar). Older versions of tar may require the ‘‘o’’ option to create the extracted files using your group ID information.

The /usr/local/lib directory now includes the umdss directory which contains the following files:

```
README                  This document in plain text format
README.mdx              Plain text MDX documentation
README.ps               This document in PostScript format
idfile.sample           Sample ID file
promail*                Mail grabbing script
promail.bsd*            Mail grabbing script for BSD
query.proline*          Sample smail query script
smroute                 Sample smail routers entry
smtrans                 Sample smail transports entry
src/                    Source code directory with Makefile
```

To build the umdss utility programs, change directories to ‘‘src’’. The source code has been written to be compatible with most popular systems (XENIX, SCO UNIX, Interactive UNIX, and BSD systems). However, some tweaks may be necessary to accomodate your system.

Before building the executables, take a look at the ‘‘config.h’’ file. You will want to read the introductory comments and make changes to it as required by your system.

When ready, issue the ‘‘make’’ command. This begins the compilation process on the source files, and if all goes well, three programs are created: umdss, mdx, and upmail. If any errors are reported, you will have to do some debugging, or enlist a guru to get it to build for your system.

To place the executables in the right spot (the parent directory), use ‘‘make install’’. To quickly remove the ‘‘.o’’ files, use ‘‘make clean’’ (or ‘‘make Clean’’ with a capital C to remove the redundant executables and object files). Then change directories to ‘‘..’’ to put you back in /usr/local/lib/umdss.

File Permissions and Directories

Make sure that /usr/local/lib/umdss and its files are owned by umdss and are in the right group with appropriate permission modes.

Now you must make the umdss spool directory. This is the directory into which outgoing mail is stored for each ProLine system (each site gets its own directory). The recommended directory is /usr/spool/umdss, should be owned by umdss and be in the uucp group. Individual site directories will be created automatically by umdss, or by the local mail system, so the directory should allow for creation of additional directories by umdss.

Verify that ‘‘idfile’’ has appropriate permissions set so that it can be read only by umdss.

Umdss creates lock files (e.g. LCK..pro-site) in the appropriate directory where UUCP stores its lock files. Make sure umdss has the ability to write to this directory.

Before you can send mail to a ProLine site, inform the UNIX mail system of the site’s existence, ensuring that the system never dials the ProLine site. This involves editing the Systems (or L.sys) file and paths database. The procedure may differ among brands of UNIX.

ProLine Site Installaton

Perform the following steps for each ProLine site that will have access to the umdss account:

Add an entry into the ‘‘idfile’’ (or edit the sample entry provided if this is your first time). The format of the idfile is as follows:

```
pro-site;password;send-command;receive-command
```

Pro-site is the name of the ProLine site (e.g. pro-apple2). Password is a security code that the ProLine site sends to verify its identity. The send-command and receive-command items are command lines that invoke the external protocol programs to exchange mail. Example:

```
pro-sol;morgan;mdx -L -s %s;mdx -L -r %s
```

This example shows that the mdx program is used for both sending and receiving. The %s characters indicate where the protocol program expects a filename. (See the ‘‘README.mdx’’ file for details on using mdx’s options).

Note: The mdx program is invoked via the C function system(). Some UNIX implementations do not include the current directory in the executable search path. You may have to add ‘‘./’’ (e.g. ‘‘./mdx’’) for mdx to be found and executed by system().

The ‘‘idfile’’ may include comments where desired. Comments follow the pound-sign (#) character.

Transporting Mail

If the UNIX host is running a smart mailer, like smail version 3.x or newer, it can be told to deposit messages directly into ProLine site directories. This is done by integrating the upmail transport program with smail. It takes a little work to get it set up, but it is far more efficient to use than any other scheme.

You should be familiar with smail's transport and routing feature. The ‘‘transports’’ file is used to tell smail how to call a transport program like upmail. (Umdss comes with a sample entry for the transports file in ‘‘smtrans’’.)

The ‘‘routers’’ file tells smail how to recognize which sites use a particular transport. Two entries are needed so that smail can locate directly connected ProLine sites as well as any ProLine sites found in a ‘‘paths.pro’’ database containing the paths to known ProLine systems. (A sample routers entry can be found in ‘‘smroute’’.)

Smail uses a query script to find out which router to use when delivering mail. (Umdss comes with a sample shell script named ‘‘query.proline’’.) By default, this script reads the umdss idfile to match directly connected ProLine systems (this means the idfile must be able to be read by smail). A more secure and more efficient scheme is to create a list of directly connected sites in the script's ‘‘for’’ loop. The drawback to this approach is that you need to remember to edit the script each time you add or remove a directly connected site.

A working example is described as follows: two routers are designated to any sites that begin with ‘‘pro-’’. They test the addresses before local and paths matching for regular traffic. The first one looks for exact matches (first hop) using the query.proline script. This script scans umdss's idfile for names that are directly connected to the host. If that fails, the next test uses a sorted paths-formatted database and lookup to resolve any path that involves a ProLine site.

(That database is made by using grep, searching for ‘‘<tab>pro-’’ in the master paths database and saving the results to paths.pro. The reason for this is so that any target ProLine site can be recognized and use the upmail transport, rather than the uux transport that would be used if there was a match in paths, which is the next test if this one fails.)

Obviously, the logic utilized to make this all work (with gateway resolution) would vary dramatically according to the mail transport. You may also need to make adjustments to the sample files if your directories differ.

Transporting Mail with Shell Scripts

If the UNIX host doesn't have a flexible mailer, like the kind described above, the umdss package includes a shell script called ‘‘promail’’ (and ‘‘promail.bsd’’ for use on BSD systems). Promail is run by either ‘‘root’’ or ‘‘uucp’’ to move mail in a ProLine site's UUCP directory (e.g., /usr/spool/uucp/pro-site) into the spool directory where umdss expects it (/usr/spool/umdss/pro-site). Control and execution files are deleted, and only the actual data files (those containing mail) are moved out of the UUCP area, into the umdss spool area for the corresponding site.

If you have to use the shell scripts, add an entry in a crontab file to execute ‘‘promail’’ (or ‘‘promail.bsd’’) as often as traffic demands. The script should be run via root or uucp, and requires at least one argument: the name of the ProLine site. Example:

```
20 * * * * /bin/su -c "/usr/umdss/promail pro-site" >/dev/null 2>&1
```

Older UNIX systems may impose a length restriction on site names. If this is the case with your system, and the site you’re adding has a name that exceeds the limit, add a second name argument: the truncated version of the ProLine site’s name. For example, if the site is called pro-apple2, and your machine imposes an eight character restriction, give ‘‘pro-appl’’ as the second argument.

Modern UNIX systems can accomodate all ProLine site names which are usually less than 12 characters, but never exceed 15.

Transporting News

The cnews program makes it easy to send uncompressed news batches through the mail system to a ProLine site. Be sure that the batches are NOT compressed and that they are sent via mail (not deposited directly into the ProLine site directories). The batches should be addressed to the ProLine site’s rnews account (e.g., pro-site!rnews).

For the ProLine Administrator

UNIX administrators will want to pass this information onto sites that will poll it:

The ProLine administrator should take appropriate steps to connect with the UNIX machine by invoking its mdss program with options that agree with the protocol being used on the UNIX system. If mdx is used with the -L (4K XMODEM) option, the ProLine site will invoke mdss with ‘‘-x TLX’’ after connecting. This tells it to enable text (newline) conversion, large 4K packets, and standard XMODEM.

A typical L.sys entry for ProLine systems might look like this:

```
unix-host Any 9600 593-7305|593-6481 local
        \[in:]2-^M;-[in:]5 umdss \[rd:] whatever
        \[id:] pro-site;password
        \!"mdss -x TLX unix-host"
```

If the ProLine system is running CS 2.7a or newer, news articles to be posted to the net should be addressed to rnews (e.g. rnews@host). Otherwise, the UNIX administrator may want to set up some aliases that redirect e-mail letters into specific newsgroups. Again, this is very easy to do with cnews.

Logfile

Umdss maintains a log file of all messages received and sent, including connect and disconnect information. (NOTE: Umdss can only create the log file if the umdss home directory is owned by umdss, or if a mdsslog file is manually created with proper permissions set).

Here is a sample section from the umdsslog file:

```
pro-sol 8/30-10:28 C i1F 38400
pro-sol 8/30-10:28 R 1636 pro-palmtree.socal.com!mporter
pro-sol 8/30-10:28 R 734 miavx1.acs.muohio.edu!tfschmidt
pro-sol 8/30-10:29 R 2041 pnet01!crash!excalibur.cb.att.com!mlg
pro-sol 8/30-10:29 S 4656 rnews
pro-sol 8/30-10:29 S 1914 mdavis
pro-sol 8/30-10:29 S 1563 rnews
pro-sol 8/30-10:29 S 3407 mdavis
pro-sol 8/30-10:29 H S#4,11540 R#3,4411 i1F 38400
```

Each entry begins with the site’s name and the current date and time. Entry codes are:

C	Connection (showing the port and speed)
R	Receive (with byte count and target address)
S	Send (with byte count and target address)
H	Hangup (showing transfer summaries, port, and speed)
*	ALERT or ERROR message
?	Debugging message

The H entry shows the number of files (and total bytes) sent after the S#, and the number of files received (and total bytes) after the R#.

The umdsslog file can grow quickly, depending on traffic, and requires occassional attention. Using grep to find lines containing ‘‘*’’ is useful for quickly spotting errors.

Jobs and Session Reports

The -q option reports current information about all ProLine sites serviced by umdss:

```
Site            Jobs Kilobytes Last Access TTY Session
=============== ==== ========= =========== === =======
pro-acsd           7     12.30 09/17-06:50 i1A success
pro-amber          4    165.12 09/19-15:55 i1D success
pro-angmar         4      7.29 09/19-00:24 i1G success
pro-beagle         0      0.00 09/19-16:14 i1D success
pro-calgary        5      8.23 09/19-02:43 i1D success
pro-carolina       6     35.99 09/19-17:25 i1D TALKING Received 5 (250K)
pro-nbs            3      7.56 09/19-14:36 i1F success
```

```
pro-nsdapple       3     21.18 09/19-14:55 i1E success
pro-party          1     25.15 09/19-16:00 i1E success
pro-sat           10    142.68 09/19-04:04 i1D success
pro-sol           34    149.33 09/19-17:19 i1M TALKING
============== ==== ========= =========== === =======
                  77    574.81
```

The -q option will report FAILED sessions as well as bogus lock (LCK) files that would otherwise indicate active sessions. Bogus locks can occur if umdss terminates abnormally, but they won't prevent subsequent sessions from starting up.

Notes

The umdss project is continually evolving as new UNIX systems implement it. Comments are appreciated on the software (and documentation) and any suggestions for making it work better.

If you make changes to the source code, be sure to enclose them within conditional directives. Do not change any of the existing code. Send your changes to mdavis@crash.cts.com so they can be included in the next official distribution.

Files

/usr/local/lib/umdss/ - umdss's home,
/usr/local/lib/umdss/idfile - site id file,
/usr/local/lib/umdss/umdsslog - logfile,
/usr/local/lib/umdss/src/ - source code directory,
/usr/spool/umdss/ - site spool directories,
/usr/spool/umdss/.Status/ - contains site's status files

See Also (ProLine)

intro(NET), poll(NET), scan(NET), sendmail(C)

See Also (Unix)

cnews(C), smail(C)

Authors

Morgan Davis (mdavis@cts.com) and Bill Blue (bblue@cts.com)

Name
unbatch - Convert rnews batches into newsgroup bundles

Syntax
unbatch

Description
Unbatch sorts the $/sys/mail/rnews mailbox into files based on newsgroup names. For example, *unbatch* might search through the rnews mailbox for all comp.sys.mac.misc articles, and writes them into a designated bundle for that newsgroup. The newsgroup bundles are stored in the $spool/news directory for later processing by *rnews*.

Unbatch should be called regularly via a crontab entry in order to process any news that came in. After the rnews mailbox is split into bundles, *unbatch* invokes *rnews* to distribute the bundles locally and/or to offsite addresses.

See *rnews* for a description of the $/etc/newsys file and its format.

Files
$/etc/newsys - newsgroup distribution file,
$spool/news - spool area for newsgroup bundles,
$/sys/mail/rnews - rnews mailbox.

See Also
ned(NET), rnews(NET), postnews(NET)

Name

uutraf - Network traffic report

Syntax

uutraf [*options*] [*logfile*]

Description

Uutraf analyzes mdss log files and provides a report on transfer statistics using a default sorting format or an optional user-specified sorting format.

By default, *uutraf* processes the current mdsslog file in the $spool/logs directory, unless an optional log file pathname is provided.

Traffic reports may be redirected into a file by adding, as the last argument on the command line, a ’>’ followed by the report file name.

Tracking Options

-p	Track only poll entries. This is useful to determine connection costs that you have initiated. Report entries are generated for each ‘‘carrier stamp’’ as given in the L.sys file.
-s	Track only by speed (baud rates).

Sorting Options

Reports can be sorted in various ways expressed below (the tag options, *r*, *x*, and *t*, correspond to *received*, *transmitted*, and *total* respectively):

-b[r|x|t]	Sort by number of bytes.
-c[r|x|t]	Sort by characters per second (CPS) rate.
-h[r|x|t]	Sort by hours
-n[r|x|t]	Sort by number of files
-a	Sort in ascending order, rather than descending.

Examples

uutraf -bt	Displays the total bytes transferred, in descending order for all active sites. This is the default reporting method.
uutraf -cx	Displays the connections with the highest characters transmitted per second rates, in descending order.

uutraf -hxa	Displays the connections by total hours of modem transmission time, in ascending order.

Note

Uutraf works on mdss log files only. Other types of log files give it indigestion.

Files

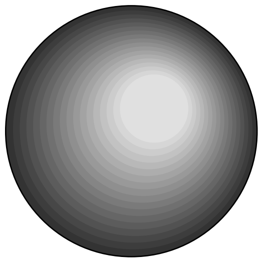
$spool/logs/mdsslog - current mdss log file.

See Also

arbitron(NET), intro(NET), mdss(NET)

ProLine

Software Development



Name

plapp - How to create ProLine applications

Description

This document describes the procedures recommended for creating applications for use in the ProLine environment.

One of the best things about ProLine is its ‘‘extendibility’’ in the form of user-written programs, and the ease with which they can be created. Not only does this provide custom functions for individual sites, but such extensions can be distributed to others, and everyone benefits. This has already been done, and the Morgan Davis Group (MDG) would like to encourage additional development along with some reasonable guidelines.

Before starting on any new or converted program for ProLine, you should have the ModemWorks Programmer’s Manual, available from MDG. This manual describes all the commands and techniques for managing the communications interface between BASIC programs and the system. MD-BASIC is also highly recommended for doing your ProLine development.

The Hard Way

Creating a ProLine program using Applesoft’s immediate input mode is possible. In fact, until 1990, ProLine was created and maintained for six years in this way. Everyone has immediate mode Applesoft, so anyone can easily create ProLine applications without any special tools -- it’s just painfully inconvenient compared to using MD-BASIC.

It also becomes difficult to maintain and support when core functionality in ProLine changes to accommodate new features. ProLine programs written in MD-BASIC using the core set of ProLine library functions need only be recompiled with little or no changes to the program source code. Programs written in Applesoft require extensive changing after analyzing the raw output from newer MD-BASIC-generated programs.

Thus, this method of development is highly discouraged since compatibility with updated versions of ProLine cannot be guaranteed nor easily maintained.

The Easy Way

MD-BASIC comes with a complete subroutine library and interfaces for creating ProLine applications in a very neat and powerful C-like environment for BASIC.

Here is a sample program for ProLine written in MD-BASIC:

```
#define IDENT_PROG "args"
#define IDENT_VERS "1.0"
#define IDENT_DATE "30mar94"
#define IDENT_NAME "Morgan_Davis"
```

```
#include <proline/proline.h>            ' Include ProLine interface

        gosub AppInit                   ' Initialize environment
        for arg = 0 to argc - 1         ' Loop through arguments
                print argv$[arg] " ";   ' Display them
        next
        print                           ' Finish up with a newline
        goto Exit                       ' And then exit

#include <proline/proline.lib>          ' Include subroutine library
```

See the *proline.h* and *proline.lib*, files for important instructions and handy constant definitions. ProLine developers can also take advantage of additional libraries, such as *launch.lib* (a simple command line processor that can be used to launch external programs and return back to yours), *getopt.lib* (a command line option processor making it easy to parse mixed arguments and flags), and more.

Resource Files

If your program includes various settings and adjustments that the user may want to configure, keep them in an external data file. Do not hard-code settings into your program. That's ugly and gross, and not very friendly. Let the administrator change the operation of your program by editing settings with an editor.

Your external data files, called ‘‘resource files’’, are stored in the $/etc/rsrc directory, and end with the *.rsrc* extension. The first part of the resource file name should match the name of your application (e.g. the ‘‘sendmail’’ application has a ‘‘sendmail.rsrc’’ file).

Do not deviate from this naming convention, as a future resource editing utility might expect to find matching ‘‘.rsrc’’ files. The format of your resource file should be fairly simple -- one data item per line if possible. Lists of items should include a count line preceding the lines of items.

If your program must store a data file for each user who uses it (e.g. user preference settings), determine if your program should allow the user to edit the file (and possibly mess up its format) if saved in the user's home directory. Preference files that are changed by your program only should be stored in the user's $/adm directory ($/adm/*username*) which is inaccessible to the user.

Input and Output

Using HOME in your program does not clear a caller's screen. Nor will commands like HTAB and VTAB -- they work only on the ProLine host's display. Refer to the &IOCTL command in the ModemWorks manual for details on local and remote screen manipulation.

Avoid Applesoft's INPUT statement for getting input, especially for numeric input. This is because Applesoft has some gross error messages that will be displayed if the user enters something it doesn't like (e.g. ?REENTER or ?EXTRA IGNORED). Plus, INPUT won't allow the user to enter certain characters. Instead, use AmperWorks' more powerful &READ statement.

When reading data from a file, use AmperWorks' &GET statement. &GET is clean. Applesoft's INPUT does stupid things like clearing the screen from the cursor to the end of the line, and then moving the cursor down one line.

Don't assume anything about the configuration of other sites. The only directories/volumes that you can reasonably assume to exist are those that are distributed with ProLine. For example, if you provide a program (or script) that makes use of ''/ram'', it will fail on any site that doesn't have a volume named ''/ram'' online!

Good Programming Tips

Here are some tips for creating good, high-performance code:

1) Use multiple statements per line. Your program executes faster and occupies less memory. For each line conserved, you save at least six bytes. (MD-BASIC automatically does this.)

2) Renumber by ones. Line numbers take up one byte per digit. The shorter your line numbers, the more you save. A good renumbering utility is recommended. (MD-BASIC automatically does this.)

3) Modularize large programs. If your application is larger than 20K in size, you should break it down into modules to leave enough RAM left for variable storage. Smaller programs run faster.

4) Eliminate unneeded processing in loops. If a statement in a loop is not necessary, move it outside of the loop. For each iteration of the loop, the statement will be executed, slowing things down.

   Bad:

```
for i = 1 to 100
    j = q * 34 / c
    r = i + j
    gosub process_r
next
```

   Good:

```
j = q * 34 / c
for i = 1 to 100
    r = i + j
    gosub process_r
next
```

5) Place frequently accessed routines at the top. Applesoft searches for program lines starting at the beginning of your program on down to the last line. The closer a referenced line is to the top, the sooner Applesoft finds it. It goes without saying: put the least commonly used routines at the bottom of your program.

6) Declare commonly used variables and constants at the start of the program. Give them short variable names (unless you're using MD-BASIC which optimizes long variable names to short ones). Applesoft can evaluate a constant value faster if it is contained within a variable.

7) Use special (sneaky) effects to hide processing delays. If the program is about to begin some lengthy number crunching, the user perceives the passage of time is faster if there is a diversion (e.g. something to read). What might pass for a friendly delay loop for reading, might actually have been the initialization of a large matrix, or maybe some disk access.

Distribution

ProLine boasts a powerful and convenient online help system that displays help documents in a programmable format for the best viewing on any screen, dot-matrix printer, or PostScript-compatible laser printer. Always include documentation in ‘‘man’’ formatted text files with your applications. See the man(CT) and manuals(M) entries for more details.

If you are distributing more than one file (e.g. the program file and its ‘‘man’’ file), place them into a ProLine archive (*par*) file. Be sure the archive retains the standard ProLine directory structure so that it can be easily unpacked on the target systems. To do this, always create ProLine archives from the perspective of the top-level ProLine System Directory ($/). Introduce new subdirectories before any files within it are referenced.

When you distribute your archive, include installation instructions (e.g. in the e-mail letter accompanying transmission of your archive, or secondary text file). This describes how to unpack your archive, be it encoded (uuencode or BinSCII) or compressed (ShrinkIt). If you can't (or won't) include installation notes, make sure the file's suffix is descriptive of what's inside:

ProLine Archive (par)
ShrinkIt Archive (shk)
Uuencoded (uu)
Uuencoded, compressed archive (shk.uu)

Steps for assembling a typical distribution:

1) Put all the files into a ProLine archive.

2) If larger than 10K, put the resulting .par file into a ShrinkIt archive (to take advantage of the compression).

3) *Uuencode* the resulting file for transmission via e-mail.

Files

$/pub/proline/plapp.exe - ProLine Application subroutines.

See Also

man(CT), manps(CT), manuals(M), par(C), updates(ADM), uuencode(C)