The Amoeba Reference Manual

User Guide

An amoeba called George and his brother
were sharing a joke with each other.
They laughed 'til they cried
and both split their sides
And now each of the boys is a mother.

1 About This Manual

1.1. Intended Audience

This manual is intended for use by all people using Amoeba, either by directly logging in or from another operating system running the Amoeba protocol.

1.2. Scope

This manual explains the basic concepts of Amoeba from a user's perspective, how to log in, configure one's working environment and how to locate required services and objects. It concludes with the collection of reference manual pages for user utilities and servers.

1.3. Advice On Use Of The Manual

It is recommended that the introductory material and tutorials be read at least once by all users. For those familiar with UNIX† the initial tutorial material may seem straight forward but the file system and directory service are different from that of conventional systems and should be understood before proceeding.

The manual pages are provided for reference about how to use the utilities and servers provided with Amoeba. (Note that some programming tools are described in greater detail in the *Programming Guide*.) It is expected that the manual pages will be used more frequently.

Throughout the manual there are references of the form *prv*(L) *ack*(U) and *bullet*(A). These refer to manual pages at the back of each guide. The following table explains what each classification refers to and in which guide the manual page can be found.

Letter	Type	Location
(A)	Administrative Program	System Administration Guide
(H)	Include file	Programming Guide
(L)	Library routine	Programming Guide
(T)	Test Program	System Administration Guide
(U)	User Utility	User Guide

There is a full index at the end of the manual providing a permuted cross-reference to all the material. It gives references to both the manual pages and the introductory material which should direct you to enough information to clarify any difficulties in understanding.

† UNIX is a Registered Trademark of AT&T Bell Laboratories.

2 Introduction

2.1. History

The Amoeba distributed operating system has been in development and use since 1981. Amoeba was originally designed and implemented at the Vrije Universiteit in Amsterdam, The Netherlands by Prof. Andrew S. Tanenbaum and two of his Ph.D. students, Sape Mullender and Robbert van Renesse. Since 1986 it has been jointly developed there and at the Centre for Mathematics and Computer Science, also in Amsterdam. It has passed through several versions, each experimenting with different file servers, network protocols and remote procedure call mechanisms. Although Amoeba has reached a point where it seems relatively stable it is still undergoing change and so it is important to take note of the various warnings and advice about the proposed design changes for future releases. This document describes Amoeba version 4.0.

2.2. The Amoeba Design Philosophy

Amoeba has been developed with several ideas in mind. The first is that computers are rapidly becoming cheaper and faster. In one decade we have progressed from many people sharing one computer to each person having their own computer. As computers continue to get cheaper it should be possible for each person to have many computers available for individual use. This is already the case to some extent.

The second relates to the widespread use and increasing performance of computer networks. The need for a uniform communication mechanism between computers that are both on a local network or on a wide-area network is already apparent for many applications.

What is needed is the ability to deal with physically distributed hardware while using logically centralized software. Amoeba allows you to connect large numbers of computers, on both local and wide-area networks, in a coherent way that is easy to use and understand.

The basic idea behind Amoeba is to provide the users with the illusion of a single powerful timesharing system, when in fact the system is implemented on a collection of machines, potentially distributed across several countries. The chief design goals were to build a distributed system that would be small, simple to use, scalable to large numbers of processors, have a degree of fault-tolerance, have very high performance, including the possibility for parallelism, and above all be usable in a way that is transparent to the users.

What is meant by *transparent* can best be illustrated by contrasting it with a network operating system, in which each machine retains its own identity. With a network operating system, each user logs into one specific machine: his‡ home machine. When a program is started, it executes on the home machine, unless the user gives an explicit command to run it elsewhere. Similarly, files are local unless a remote file system is explicitly mounted or files are explicitly copied. In short, the user is clearly aware that multiple independent computers exist, and must deal with them explicitly.

By contrast, in a transparent distributed system, users effectively log into the system as a whole, and not to any specific machine. When a program is run, the system, not the user,

‡ ‘‘his’’ will be used in the generic sense to avoid writing ‘‘his or her’’.

decides the best place to run it. The user is not even aware of this choice. Finally, there is a single, system-wide object naming service. The files in a single directory may be located on different machines possibly in different countries. There is no concept of file transfer, uploading or downloading from servers, or mounting remote file systems. The fact that a file is remote is not visible at all.

2.3. The System Architecture

The Amoeba architecture consists of four principal components, as shown in figure 2.1. First are the workstations, one per user, on which users can carry out editing and other tasks that require fast interactive response. The workstations are all diskless, and are primarily used as intelligent terminals that do window management, rather than as computers for running complex user programs. We are currently using SUN-3s and VAXstations as workstations. In the next generation of hardware we might use X-terminals as well.

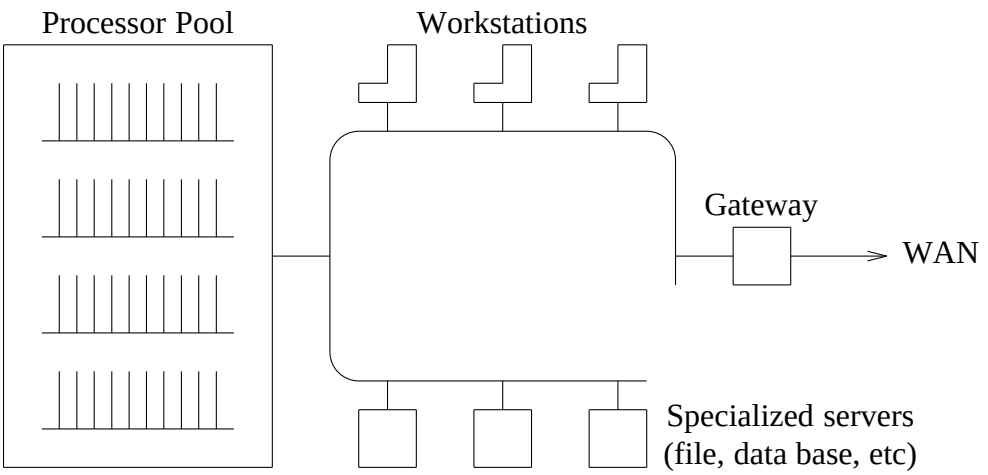

Fig. 2.1. The Amoeba architecture.

Second are the pool processors: a group of CPUs that can be dynamically allocated as needed, used, and then returned to the pool. This makes it possible to take advantage of parallelism within a job. For example, the *make* command might need to do six compilations, so six processors could be taken out of the pool for the time necessary to do the compilation and then returned. Many applications, such as heuristic search in AI applications (e.g., playing chess), use large numbers of pool processors to do their computing. The processor pool also offers the possibility of doing many jobs in parallel (eg. several large text processing jobs and program compilations) without affecting the perceived performance of the system because new work will be assigned to idle processors (or the most lightly loaded ones).

Third are the specialized servers, such as directory servers, file servers, boot servers, and various other servers with specialized functions. Each server is dedicated to performing a specific function. In some cases, there are multiple servers that provide the same function, for example, as part of the replicated file system.

Fourth are the gateways, which are used to link Amoeba systems at different sites and

different countries into a single, uniform system. The gateways isolate Amoeba from the peculiarities of the protocols that must be used over the wide-area networks.

All the Amoeba machines run the same kernel, which primarily provides multi-threaded processes, communication services, and little else. The basic idea behind the kernel was to keep it small, to enhance its reliability, and to allow as much as possible of the operating system to run as user processes (i.e., outside the kernel), providing for flexibility and experimentation. This approach is sometimes called a *microkernel*.

2.4. Objects and Capabilities

Amoeba is object-based. It uses capabilities for accessing objects. The Amoeba system can be viewed as a collection of objects, on each of which there is a set of operations that can be performed. For example, typical operations for a file object are creating, reading and deleting. The list of allowed operations is defined by the person who designs the object and who writes the code to implement it. Both hardware and software objects exist.

Associated with each object is a *capability*, a kind of ticket or key that allows the holder of the capability to perform some (not necessarily all) operations on that object. For example, a user process might have a capability for a file that permitted it to read the file, but not to modify it. Capabilities are protected cryptographically to prevent users from tampering with them.

Each user process owns some collection of capabilities, which together define the set of objects it may access and the type of operations it may perform on each. Thus capabilities provide a unified mechanism for naming, accessing and protecting objects. From the user's perspective, the function of the operating system is to create an environment in which objects can be created and manipulated in a protected way.

2.5. Remote Procedure Call

This object-based model visible to the users is *implemented* using remote procedure call. Associated with each object is a *server* process that manages the object. When a user process wants to perform an operation on an object, it sends a request message to the server that manages the object. The message contains the capability for the object, a specification of the operation to be performed, and any parameters required for the operation. The user, known as the *client*, then blocks. After the server has performed the operation, it sends back a reply message that unblocks the client. The combination of sending a request message, blocking, and accepting a reply message forms the remote procedure call, which can be encapsulated using stub routines, to make the entire remote operation look like a local procedure call.

A remote procedure call can impact perceived performance because it is possible for it to fail when it cannot find a server (temporarily or otherwise). Although this is a rare occurrence, it does mean that sometimes there may be a brief delay while a program attempts to locate a server. The delay may vary between two and thirty seconds. (Of course it is initially difficult to distinguish between a long timeout and someone rebooting the machine where your process is running, but that is a separate issue.)

2.6. Wide Area Networking

In general you will be unaware of the existence of wide-area networking. User programs that are accessing remote servers will not be aware of it and the only evidence of wide-area network activity for the user will be delays if the wide-area network is slow (which they typically are).

Communication over a wide-area network uses the network protocol of the wide-area network. This is hidden from the user who does remote procedure calls in exactly the same was as for local-area communication. Special gateway software arranges that the messages arrive at the recipient in such a way as to appear identical to local messages.

Wide-area networking is not yet fully supported in Amoeba 4.0, but will be in subsequent releases.

2.7. Directories

Under Amoeba files and file names are not implemented by the same server. This is because in Amoeba there are many more object types than just files and a more general naming service is required. The naming service is implemented by the directory server which implements an arbitrary directed graph of directories. This graph can contain cycles and does so in the standard distribution. This is fully intentional and causes no problems. The directory object is a list of (ASCII string, capability set) pairs. The *ASCII string* is the human readable name of the object and the *capability set* contains one or more capabilities for the various copies of that object. (This provides support for fault tolerance by allowing the same name to refer to the replicated copies of an object.)

The capability for any object, including another directory, may be stored in a directory, so a directory can be linked into several places in the directory graph simply by entering its capability in several directories.

The directory graph has no concept of an absolute *root* since there is a no tree structure. However the user interface implements the root for a particular user as the user's login directory. Therefore all users have their own '/' directory.

An extra degree of fault tolerance is achieved by duplicating the directory server. Two copies of it can run simultaneously, each with its own set of data. The two servers communicate with each other to keep the data about directories consistent. If one server goes down, the other detects this and carries on functioning. When its partner is restarted it is brought up to date with the latest changes by the surviving server and they resume cooperation.

This provides a very high reliability in the directory service and allows software upgrades to the directory server without down time by simply stopping the servers one at a time and starting the new version.

2.8. Files

Files are also unusual in Amoeba. The file server delivered with Amoeba is known as the Bullet Server. It uses a very large buffer cache and stores files contiguously, both in core and on disk to give very high performance.

The semantics of Bullet Server files are different from conventional file systems. The foremost difference is that they are immutable. That is, once created a file cannot be

changed. The only significant operations permitted on a file are read and delete. (There are some operations which deal with the status information of the file.)

The second aspect of files is that they are created atomically. They can be constructed in the Bullet Server's buffer cache but they do not officially exist until they are complete, at which point they are *committed* which causes them to come into existence and be written to disk.

2.9. Amoeba and UNIX Emulation

To provide Amoeba with a reasonable programming environment and to simplify the migration of software and staff from UNIX systems to Amoeba, there is a UNIX emulation library which provides reasonable source code compatibility. A special server handles state information when necessary, but this is not explicitly visible to the user. It is started when required.

Functional equivalents of many of the standard UNIX tools and utilities are provided with Amoeba. In practice this means that the user environment of Amoeba is strikingly like that of UNIX, even though no AT&T code is present. However the directory service and file semantics are significantly different so it is important not to take the similarity with UNIX for granted. For example, commands like `tail -f` are not implementable since file creation is atomic.

2.10. Conclusion

Amoeba was designed to provide a transparent distributed system. Although it is still undergoing further development the current version goes a long way towards satisfying the design goals.

The UNIX emulation has provided a ready-made programming environment for people to begin using Amoeba immediately. To help get started the next section provides a tutorial introduction to using Amoeba and highlights some of the areas where Amoeba differs from UNIX.

3 Getting Started

This is a tutorial introduction for beginners which gives the basic information about how to log in and out, start up the window server, use some simple tools and navigate through the directory structure. The first section on logging in and out is intended to get you started quickly. It has been written with total beginners in mind.

3.1. Logging In and Out

Before you can use Amoeba you will need to go to your system administrator and request a *user name*. This is the name by which you are known to the computer system. When you are allocated this name, resources will be set aside in the system for your use. You use this name to gain access to these and other system resources.

Amoeba uses passwords to prevent unauthorized use of users' accounts. When you are assigned a user name you must also give a password to go with the name. You should keep your password secret.

It is important to choose a password that will be difficult for others to guess. Names of wives, husbands, children, girl friends or boy friends are extremely bad choices of password. So are words from English or the local spoken language. Make sure, however, that the password you choose is not too difficult to remember. There is a program to change your password as well and it should be used regularly to ensure security.

When you want to use Amoeba it will require you to give your user name and password before permitting you to access the system (unless someone is already using the workstation). This is known as *logging in*.

To use Amoeba you need a workstation. When you sit down at a workstation you will see a short text followed by the word `Username` somewhere on the screen. For example:

```
Amoeba 4.0 release 1.

Username:
```

The system is asking you to identify yourself by typing in the user name assigned to you by the system administrator. Therefore in response to this you should type in your user name and then press the key labeled *return*. In response to this the computer will print:

```
Password:
```

You should type in your password. The computer will not show it on the screen as you type it. Once you have typed your password you should once again press the *return* key.

If you typed your password incorrectly it will print the message

```
password incorrect
```

and you will get the chance to try again. If you receive any other error message then consult your system administrator.

If you typed your password correctly then a command interpreter will be started for you. A command interpreter is a program that accepts the commands that you type at the keyboard and executes them for you. It will print a *prompt* to indicate that it is ready for you to enter commands. The prompt from the default command interpreter is

```
Amoeba>
```

All the commands that you type to the command interpreter are only executed after you press the *return* key. This is a signal to the computer that the whole command has been typed. In response to this you should type the command

```
echo hello
```

Do not forget to press the *return* key after you type the two words. The system should respond by printing

```
hello
```

Another command that you can try is *date*. This should print the Amoeba's idea of the current time.

As important as getting started and typing commands is knowing how to stop using the system. This is quite simple and is known as *logging out*. In response to the prompt

```
Amoeba>
```

you simply hold down the key labeled *control* and press the key for the character *D*. (It is not necessary to press the return key in this case!) The system may print a new line and then once again ask you for your user name.

3.2. Creating, Modifying and Destroying Files

As with most other computer systems, Amoeba provides files for the storage of data. A file is just an array of bytes (in the UNIX tradition). The file server distributed with Amoeba is called the Bullet Server. Users can create and modify files in many ways. Amoeba comes with several text editors for creating and modifying human readable text. Manual pages describe each editor and there is a section later in this manual containing the reference manuals for the editors. There are three editors currently available. The first is called *elvis*(U). This is an imitation of the Berkeley *vi* editor, but provides some extra features as well. It has been linked to the names *vi* and *ex* under Amoeba. There is also an editor called *be*(U) which has an *Emacs* flavor and an editor called *ed*(U) which is very similar to the UNIX *ed* editor.

Files are removed using the command *del*(U). To actually destroy the object it is necessary to use the **–f** flag of *del*. You should only use the **–f** flag if you are sure that nobody else has a link to the file or that you wish to deny further access to the information in it. At present there are no reference counts kept for files so a garbage collection mechanism will destroy any files no longer referred to by any links.

The UNIX emulation also provides the *rm* command, but this does not destroy the file, even with the **–f** flag. It simply removes the directory entry.

3.3. Customizing Your Environment

As with many other systems there are many aspects of your environment that can be customized. A very important choice for many people is that of command language interpreter. The command language interpreter is known as the *shell*. At present there is only the MINIX shell available. It is a clone of the UNIX No doubt more shells will be ported to Amoeba in the near future and we are planning to develop a native Amoeba shell.

There are two types of environment variables under Amoeba. They are given to each process by the creator of the process and after the process starts they can only be modified by the process itself. The first is the traditional string environment provided by many operating systems. This provides a mapping from a string (the environment variable name) to an arbitrary string. Many programs use this to have a fixed way of referring to a file by referring to the variable name. The variable in turn contains the name of the file to use. The second is the capability environment. This has a similar function to the string environment but maps the environment variable name to a capability. This is exactly the function of a directory (ie., mapping a name to a capability), but is found inside the process' data and is not implemented with the semantics of a full directory server.

The simplest way to set up your initial environment is to modify the file called *Environ* in the directory ''/''. This file contains your encrypted password, initial shell and items to add to your capability and string environments. It is executed once when you log in and sets the initial environments for your shell.

The first line of the file contains your password. This you can set using the program *chpw*. The other lines of the file can be modified using a text editor. (Be careful not to modify the password line of the file.) The syntax for lines to set string environment variables is:

```
setenv VARIABLE value
```

The syntax to set capability environment variables is:

```
setcap VARIABLE pathname
```

where *pathname* is a path in the directory server which can be looked up to return a capability to insert into the environment. (NB. The path is not put into the environment!) The initial shell is specified by the line

```
exec command1 command2
```

Because the MINIX shell needs UNIX emulation you need to first start a session server because the shell does not start it for you automatically. Therefore you should have the following lines in your environment file:

```
setenv SHELL /bin/sh
exec /public/server/session /bin/sh
```

In addition to the *Environ* file many shells also have their own start up file(s) for configuring the environment. The *Bourne Shell* clone provided with Amoeba uses the file called *.profile* which it looks for in your home directory (as specified by the environment variable `HOME` which should be defined in your *Environ* file).

3.4. Starting X Windows

If the system you are using has workstations with bitmapped displays, such as a Sun 3/60 or an X terminal, it is possible to use the X11 Release 4 window system. The binaries for X windows are normally found in the directory */public/module/x11/bin*. You should begin by putting this in your path. Should you desire to do this the following steps are required. If you are using a workstation running Amoeba you should ensure that the workstation is running an Amoeba kernel with the necessary drivers for X windows. These are in the normal workstation kernel. Then it is necessary to start an X server on your workstation.

The system administrator can arrange for this to happen automatically when your system is rebooted.

If you wish to use an X terminal then when it starts up we assume it is already running an X server. It is necessary to have a TCP/IP server running on an Amoeba host when using an X terminal. The system administrator must arrange this.

The last step required is that the system administrator arrange that a login process (known as *allo*(A)) be started which communicates with the X server on your workstation. This will prompt you for your login name and password as previously described in the section on logging in.

Once you have logged in it is possible to start a window manager and create windows running on Amoeba. It is also possible to open windows where the processes run on a UNIX system if there is a TCP/IP server available. See the section *Remote Login* later in this manual for details.

The best place from which to start a window manager and some windows is in your *.profile* file. This is normally kept in your login directory. This is a command script executed by your shell when you log in. Below is an example of a *.profile* file for starting X windows. For further details it is recommended that you consult the X11 Release 4 documentation and/or an experienced X windows user.

```
PS1="Amoeba> "
PATH=.:/bin:/public/util:/public/module/x11/bin:
SPMASK=0xff:0x2:0x4
TCPIP_SERVER=/public/cap/tcpipsvr/tcpip1
export PATH PS1 SPMASK TCPIP_SERVER

if [ X$DISPLAY != X ]
then
          TERM=aterm
          export TERM
          XPATH=/public/module/x11/bin
          $XPATH/xrdb -nocpp $HOME/.Xresources
          $XPATH/twm &
          sleep 4
          $XPATH/xclock -display $DISPLAY -geometry 80x80-2+2 -analog &
          $XPATH/xload -display $DISPLAY -geometry 140x80-100+2 &
else
          echo -n 'TERM=(sun)'
          read answer
          case $answer in
          "") TERM=sun;;
          *) TERM=$answer;;
          esac
          export TERM
fi
```

When you log in on a machine running an X server it is customary to start an *aterm*(U) window on it. This is the Amoeba equivalent of *xterm*. A shell will run in it and this shell

will execute your *.profile*. The first few lines of the *.profile* will set some important string environment variables. It is important to note that if the variable `TCPIP_SERVER` is not set then you will use the default TCP/IP server. The `PATH` environment variable tells the shell where to look for the commands that it should execute. The `PS1` variable describes what the primary command prompt should look like.

The *if* statement is checking to see if you are using a workstation which is running an X server. If not then it enquires whether the terminal type is a *sun*. If it is just press return. (You should replace the type *sun* with the terminal you most commonly use.) Otherwise fill in the name of the terminal type as described in the file */etc/termcap*. If you are unsure about this information consult your system administrator.

If there is an X server running then the terminal type is set to *aterm.* Then the X resources data base is constructed, the *twm* window manager started and then two simple X utilities are started.

3.5. Directories

The directory server provides a way of naming objects. It does this by mapping human readable strings to capabilities. The name used to reference an object is known as a *path name*. Under Amoeba your login directory has the name ‘‘/’’ (pronounced slash). All other directory names are ASCII strings not containing the character ‘‘/’’ and a path name is simply a list of directory names separated by slashes. There are also several programs for examining the directory structure and adding and deleting directories.

If a path name begins with ‘‘/’’ then it is absolute from your login directory. An example of an absolute path name is:

```
/public/cap/bulletsvr/default
```

Using the shell provided with Amoeba it is possible to move about within the directory structure and the system will remember the directory you are currently in. If a path name does not begin with a ‘‘/’’ then it is interpreted as being relative to the current directory.

Unlike many other directory systems, the Amoeba directory server does not enforce a hierarchical tree structure. It allows an arbitrary graph with cycles and cross links. To assist in the navigation through the source tree there are two special directory names defined. The first is ‘‘.’’ (pronounced dot) and refers to the current directory. The second is ‘‘..’’ (pronounced dot dot) and refers to the previous directory in the current path name. The names ‘‘.’’ and ‘‘..’’ cannot be referred to as real directories but are interpreted by the directory path look-up software. Attempts to modify ‘‘.’’ or ‘‘..’’ entries will fail. (Note that this avoids the problem with the UNIX definition of ‘‘.’’ and ‘‘..’’ which do not return along symbolic links.)

3.5.1. The Soap Directory Server

The directory server distributed with Amoeba is described below. It is known as *Soap*.

A directory is a two-dimensional matrix. Each row and column is labeled by an ASCII string. Within a directory, all row and column names are unique. Also associated with each row is a set of capabilities. Each row maps its ASCII name to the set of capabilities in that row. The columns describe the access rights of the different groups that have access to the objects.

For example, a directory could be set up with three columns: the owner, the owner's group, and others, as in UNIX. The directory shown below handles the binding for two files, *README* and *x.c*, and a directory *public*.

name	cap set	*owner*	*group*	*others*
README	file 1	111	100	100
public	directory	111	110	100
x.c	file 2	110	100	000

The owner of the directory has defined three UNIX-like access groups: the owner, group members, and others with whom the owner may share these objects. Each entry contains a set of three rights mask bits, for example, for read, write, and delete respectively. In our example, users that are in the same group as the owner of this directory are not allowed to write or delete the file *x.c*. Others are not allowed any access to this file. To prevent accidents, the owner has also turned off the delete right.

Although an object pointed to by the *owner*, *group*, and *other* columns is one and the same object, the capabilities for them are not, because they have different access rights. Thus these capabilities will have different Rights and Check fields.

A capability for a directory specifies a particular column (or columns). For example, a capability which is only good for *group* members will not return capabilities with all the rights bits on, but will restrict them according to the rights mask in the *group* column. The directory service can generate capabilities for the various columns on-the-fly as needed, but it can also decide to cache them, at its discretion.

When a new version of an object is available, the owner presents the directory service with a new (name, capability) pair. This action invalidates all the old replicas, which the directory service deletes by sending messages to the appropriate services. The directory service also creates new replicas. All of this work is done automatically by the directory service.

3.5.1.1. Operations on Directories

The bits in the *Rights Field* in a capability for a directory are defined as follows

bits 0-3 These bits tell which columns in the directory may be accessed for look-up. Bit 0 is set if the first column may be accessed, bit 1 if the second column may be accessed, etc. to a maximum of six columns. The rights mask to be used for creating the capability in a look-up operation is formed by OR-ing together the masks in the accessible columns.

bits 4-5 These bits are reserved for future use.

bit 6 This bit gives the right to modify the directory, that is, adding, deleting and changing rows in the directory.

bit 7 This bit gives the right to delete a directory. For safety only empty directories may be removed.

Note that other directory operations, such as listing the names of rows and columns, are available to any holder of a capability. They are not protected by rights bits.

3.5.1.2. Commands for Soap

For the user there are six basic commands for manipulating, and getting information from, directories:

dir list the contents of a directory.

del delete an entry in a directory and optionally the associated object.

mkd create a new directory.

chm change the mode of a directory entry.

get get the capability set associated with a directory entry and write it on standard output.

put takes a capability set from standard input and create the named directory entry.

These commands are described in the manual pages of this volume. It is worth learning how they differ from the UNIX equivalents provided.

3.5.2. The Directory Structure

Although there may be some variation from site to site, the graph in figure 3.1 shows the standard directory structure for Amoeba, starting from the root directory for user *Tom*. This should provide some idea of how to find things.

Remember that the root directory for each user is his login directory. Also note that the link `/public` restricts rights of access so `/public/users/tom` will have fewer rights than the directory `/`.

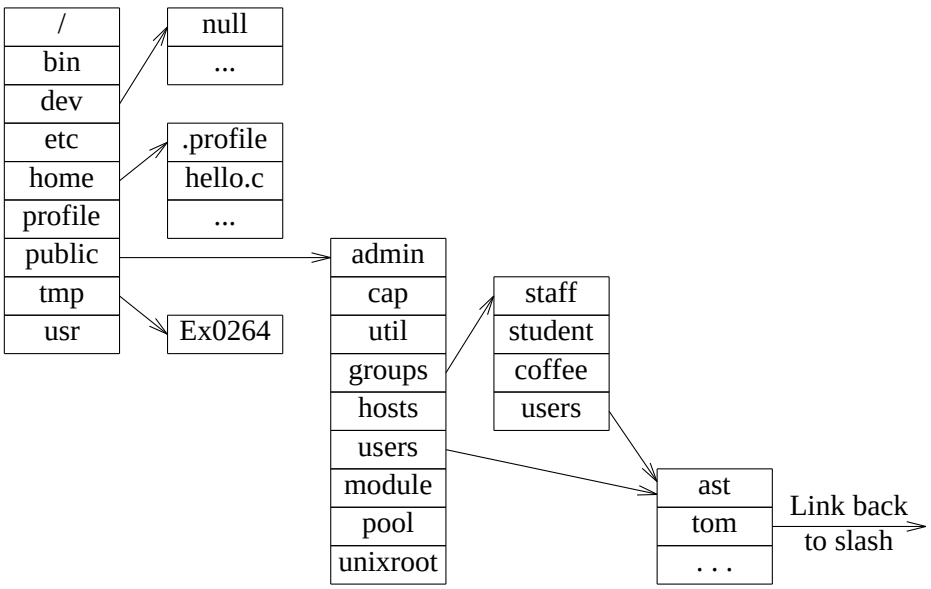

Fig. 3.1. The Directory Structure as seen by user *Tom*

3.6. Tools and Languages

Amoeba provides several programming tools and languages. Besides many of the standard tools from UNIX, such as *yacc* and *awk*, we have a version management system for building programs and keeping them up to date. This program is called *amake*(U). For most uses all that is necessary is to include the standard tools and provide a list of source files and the name of the object to be made. All the dependency information is deduced and checked by *amake*.

Amoeba is distributed with compilers made using the Amsterdam Compiler Kit (ACK) which is described in some detail in the programming guide. It provides several languages and can generate code for many different kinds of machine. It can be used for cross-compiling when porting code to new systems. The programming languages C, Pascal and Modula 2 are provided. In addition assemblers are provided for several architectures. Amoeba also provides a new language called Orca which was specifically designed for parallel programming. How to use these tools is described briefly in the manual pages and in detail in the *Programming Guide*.

How to compile the C program in the above edit session.

3.7. On-line Help

Presently the only on-line help is the command *man*(U). It provides a very simple synopsis of the major commands. In the next release it should be possible to use the more sophisticated manual system available with X windows.

4 Advanced Users

4.1. The Pool Processors

The Amoeba pool processes are usually allocated dynamically as needed. A special server (the *run*(A) server) knows about the set of pool processors available to each user. It allocates processors for a user only from his pool. This consists of the processors in the subdirectories of */profile/pool*. The *run* server regularly collects information about the amount of free memory and the CPU load of each pool processor. It uses this to select the best processor to run a process on. If the *run* server is not running then the processor selected will be on the basis of round-robin through the processors in */profile/pool*. This is particularly slow since the attempt to use the *run* server must time-out before it resorts to the round-robin strategy. If command startup suddenly seems very slow then the run server may not be present.

It is possible to have more than one subdirectory in */profile/pool*. For example, it is possible to have a subdirectory per architecture type. If the run server is aware of all these pool directories, all the machines in all directories will form your processor pool. At present however the *run* server does not know how to handle multiple architecture types so it is safer to have just a single architecture subdirectory. In the next release the *run* server will know how to select the right architecture for a particular binary that has to be executed.

Since you do not normally need to know where your processes are running your are not told. To find out what processes you are running and where you can use the *aps*(U) command. This will tell you something of the status of your processes. Currently it tells you on which pool processor each process is running and whether it is running or not. The user name that it prints with each process is merely a comment added when the process is started. It is not to be taken at face value since it is not really possible to determine the true owner.

The command *stun*(U) will allow you to kill processes that are not accessible by the interrupt character from your terminal. It is much more powerful that the *kill*(U) command in the UNIX-emulation and correspondingly more dangerous. It is very easy to kill someone else's process by mistake. This problem will be resolved shortly.

4.2. The Unix Emulation

Amoeba provides versions of many of the utilities normally available under UNIX. These are normally found in */bin*. Native Amoeba utilities are normally found in */public/util*. In some cases there are UNIX-emulation utilities with the same functionality as native Amoeba utilities but they are provided for people familiar with UNIX so simplify getting started with Amoeba.

To implement things like shared file descriptors between processes, pipe-lines and */dev/tty* there is a server known as the session server (see *session*(U)). Usually this is started when your login shell is started. Only one is needed per user and you are not normally aware of its presence. However if you start up a lot of processes that want to do a lot of I/O then the session server is a potential performance bottleneck.

At first glance Amoeba's UNIX emulation gives the illusion that you are using a truly UNIX-like system. It is exactly that. An illusion! There are several aspects of Amoeba which are subtly incompatible with the usual UNIX environment. It is important to understand these differences to avoid making costly and unpleasant mistakes. Most of these relate to

incompatibilities due to the Soap Directory Server and the immutable files of the Bullet File Server. Note that the ASCII name of an object is not in the same server as the object. A server identifies objects using capabilities, not names. For example, under Amoeba, files may be stored in the Bullet Server, while ASCII names for files are stored in the Soap Directory Server. Under UNIX The file service and the name service are tightly intertwined. This difference between UNIX and Amoeba is necessary since the name server provides names for more types of object than just files.

One major difference to bear in mind is that the directory structure is an arbitrary directed graph with cycles. Therefore commands such as

```
rm -rf foo
```

are dangerous if the directory *foo* has a subgraph containing cycles, or worse yet contains a link to some point in the graph which leads back to *foo*.

The next major difference concerns links. There are no symbolic links in the Soap Directory Server. The only kind of links are hard links but they have a different semantics from those of UNIX hard links. The directory server stores (name, capability set) pairs. This means that if you store the capability for an object under a the name *jack* and then run the command

```
ln jack jill
```

then the directory server stores the capability in two places. It does not keep link counts. If you now replace the capability *jack* with a new capability, the entry for *jill* will still have the original capability that it stored when the *ln* command was run. Thus the link is broken. These semantics are important to bear in mind when deleting objects. If you execute the command

```
del -f jack
```

this will tell the server to destroy the object *jack*. The server will simply comply and destroy the object (assuming that you have sufficient rights), regardless of whether or not anyone else holds the capability for this object. There are no link counts anywhere so the object vanishes, breaking any links. It is possible that the semantics of the links can be altered in a later release of Amoeba so that they are like those of UNIX.

The Bullet Server implements immutable files. That is, once a Bullet file has been created it cannot be changed. If you edit a file, then the file is first copied, then modified and an entirely new file is created. The new file has a different capability from the original. This new capability is normally stored under the same name as the original file and the original file is deleted. This results in breaking any links to a file whenever it is modified for the reasons explained in the previous paragraph.

These differences will normally not be a problem unless the UNIX semantics for links are required.

Since capabilities are used for protection of objects, Amoeba does not need or have the UNIX equivalent of user ids or group ids. This means that programs that rely on user or group ids will not work in the same way under Amoeba, if they are present at all.

UNIX only knows about one type of object, namely the *file*. Since Amoeba knows about many more types of objects, programs such as *ls*(U) can get confused about what protection modes and file types to use for some objects, since they try to interpret everything as a file or directory. In particular, if a capability for an object that no longer exists, or whose server is

down, is stored in a directory then *ls* will sometimes describe this directory entry as being for a character special device. To get accurate information about the state of a directory in this case use the native Amoeba command *dir*(U).

In general these differences between UNIX and Amoeba should not present many problems as long as they are kept in mind.

5 Editors and Text Processing

There are several editors available under Amoeba. In particular there is the *vi* clone *elvis*. Furthermore, this editor actually provides more functionality than *vi/ex* and a the interface is subtly different but the differences should not affect many people. There is also an *ed* clone.

The screen editor *be*(U) is provided. It is patterned after *Emacs*, but is simpler. The reference manuals for these editors are provided below.

A version of the stream editor sed (see *sed*(U)) and awk (see *bawk*(U)) have also been provided. See the *Tools* section of the *Programming Guide* and the manual pages in this volume for further information.

Currently the only text formatter available is *nroff*(U) from MINIX. It is hoped that TeX will soon be ported to Amoeba. Unfortunately it is not possible to distribute *troff* due to licensing restrictions. Sites with the appropriate source license for *troff* should be able to port it to Amoeba.

5.1. The BE Editor

5.1.1. Introduction

Be is a general text editor which is intended to be easy to learn and easy to use for novices, yet fast and powerful for experts. It borrows ideas from several sources, notably Emacs and the Apple Macintosh, but the combination of features is thought to be unique, and it is believed that *be* compares favorably to either. *Be* is in no way a fixed product; in fact, it is only a stage underway in the construction of a language-dedicated programming environment for the B programming language (hence the name).

5.1.1.1. About This Manual

If you are not a very experienced UNIX or Amoeba user you may like to ask someone to set things up properly for you. However by careful reading of this manual it should be possible to do it yourself.

There is a lot to say about any editor. In fact, there is usually so much to say that a single manual cannot serve both novice and expert users, or both comprise an introduction and a reference manual. This manual is intended as an introduction to editing with *be* for fairly novice users, but it cannot be helped that if you have never used UNIX-like systems before you will need some help from a more knowledgeable friend when first using *be*.. Since writing a reference manual would be premature in the current state of development *be*, experts will have to skim through the introduction in search for interesting features. For reference, an exhaustive (but cursory) list of command descriptions is available.

There is some information that is likely to differ between computer installations or change in time due to administrative policies, such as the command you have to type to enter the editor, and instructions for how to tell it what kind of terminal you are using. This information is given here for *be*'s ''home machine,'' but these instructions may not apply to your machine. Finally, there is information that depends on the kind of terminal you are using; see the section *Keys and Commands* below for a word on this subject.

5.1.1.2. Disclaimer

This manual describes the version of *be* which runs on computers running some version of the UNIX or Amoeba operating systems. In fact it has only been tested under UNIX version 4.2 BSD and Amoeba 4.0, and some of its features critically depend on features available in those particular systems. These dependencies are mentioned in the text, insofar as they are known.

Since *be* is still under development, no guarantee is given that this manual is complete or even correct, although most changes will not affect existing commands, or affect them in a systematic, predictable way.

5.1.1.3. Keys and Commands

This document is generic for all types of terminals. It mentions commands like [backspace], [up] or [undo] that refer to ''virtual keys'' that are not present on all terminals. Another document should tell you how to type these. If you have a terminal with many function keys, most commands may be invoked by pressing a function key; otherwise you may have to type several keys to get a specific command, or use the control key. *Be* has default ways of typing

these command that work on most terminals, but unless you are already used to Emacs you are not encouraged to use these, as they are hard to remember.

An appendix gives the bindings used locally on Ampex 230 terminals, and the default bindings are easily discovered with the use of the [help] command, which is itself by default bound to ESC-ESC and ESC-?. For example, to find out the binding(s) for [copy], type ESC-ESC, and then (when the cursor has moved to the bottom line saying ‘‘Help search string:’’), type ‘‘copy’’ and press the [return] key. Incidentally, [help] displays the actual bindings, as affected by any **–b** options. Unfortunately, when bindings have been prepared to match a certain terminal’s function keys, [help] displays the character sequences transmitted by the function keys, not their names.

Note that the labels stuck on function keys are usually not the full command names but suitable abbreviations; this should cause no problems in practice since the meaning is clear enough.

5.1.1.4. Entering BE

Before you can use *be*, you must first somehow start the computer program of that name. You have to type the command *be* in lower case. To edit a file named `foo` with *be*, type to the shell:

```
be foo
```

You may want to make *be* your default editor for editing mail and other things. To do this, set the environment variable EDITOR to the full path name of the *be* program, and export it.

5.1.1.5. Command Line Options

There is a manual page for *be* in the User Guide (see *be*(U)). It describes the various command line options in some detail.

5.1.2. Typing; Making Small Corrections.

When you enter the editor, the terminal’s screen will be cleared. The bottom two lines contain some messages that you can ignore for now, and the terminal’s cursor appears in the upper left corner. (If you start editing an existing file, the first portion of that file is displayed in the empty space.) Any characters you now type will be inserted on the screen at the position of the cursor, which of course moves along with what you type. If you press [return], the cursor goes to a new line and what you type now appears there.

If you type more than the screen width (usually 80 characters) without pressing [return], the characters will go to the next line, but this is really a feature of the display: in the file, the characters still form one very long line. So remember to press [return] or set the right margin if you want to have control over the length of the lines; for instance, printers often ignore excess characters on a line.

If you type a wrong character, you can use the [backspace] key to erase it. The character to the left of the cursor is erased and the cursor moves one place back. You can erase more characters by repeating [backspace]. If the cursor is at the begin of the line, [backspace] brings you to the end of the previous line (if there is one; otherwise it does nothing).

5.1.3. Moving Around; More Corrections

If you discover a mistake a few words back, it would be annoying to have to backspace all the way back and start retyping from there (which is essentially what you have to do if you are not using an editor). Instead, you can move the cursor backward without erasing characters by pressing [left]. Each time you press it the cursor moves one position to the left. When the cursor is just after the wrong character, you can press [backspace] to erase it (the rest of the line moves one position to the left to fill the left open space) and then type the replacement (the rest of the line moves right again). You can of course also only erase one or more characters or only insert one or more characters. To move right again, use [right].

When you are at the beginning of a line, [left] moves to the end of the previous line. When you are past the last character of a line, [right] moves to the beginning of the next line.

There are other commands to move around in the file: [up] moves the cursor to the previous line, remaining in the same column if possible; [down] similarly moves to the next line. For longer movements within a line, there are commands [word-left] and [word-right] that move to the previous or next space, and the commands [begin-of-line] and [end-of-line] move directly to the beginning or end of the current line. For longer vertical movements, there are commands [scroll-backward] and [scroll-forward] that move by almost as many lines as fit on the screen (in fact slightly less, to provide some feedback as to where you end up). Finally, there are commands [begin-of-file] and [end-of-file] that take you directly to the beginning or end of the file.

It may seem difficult to remember all these commands; in fact you probably will not need most of them for a while, as you can move around freely with only [up], [down], [left] and [right]. When you get more familiar with the editor, you can start using the others to your advantage.

5.1.4. How to [undo] Unwanted Changes

Be was designed to remove the fear of losing your work. For instance, when you exit from it, it will automatically save all files that you have changed: you do not need to give an explicit ‘‘save’’ command. There are, however, cases where you have accidentally erased more than you wanted from a file, or otherwise botched your work (maybe the cat walked over your keyboard at home).

At any moment, when you have pressed a key, you can undo its effect (i.e., pretend you never pressed it) by pressing [undo]. If you have pressed several keys that you want to undo, you can press [undo] several times, until all unwanted effects are gone (with a maximum of 40 steps).

Sometimes, one press of [undo] undoes a sequence of several similar keys; this is the case with ordinary characters (so that typing a whole line can be undone in one key stroke; correcting single characters is more easily done with [backspace] anyway), and with series of small motion commands like [up], [down], [left] or [right].

If you decide you did not really want to undo a command, or you pressed [undo] too often, there is also a reverse command: [redo]. You can see [undo] and [redo] as commands that move you along a time axis: [undo] moves you one step back, while [redo] moves you forward. You cannot [redo] if the previous command was not [undo], however (after cancellation of matching [undo]-[redo] pairs).

To be fair, there is one situation where your work is not saved: when the computer crashes, the editor dies without being capable of saving your work. However, *be* does make an attempt to prevent crashes from being too disastrous. Consult the section on backups for information.

5.1.5. The Focus; [cut] and [paste]

Like the Apple Macintosh, *be* is capable of displaying a *focus*; that is, a highlighted portion of the text. Creating a focus is slightly more complicated than on the Macintosh, but once you have created it you can do the same things to it and more. The highlighted text appears in inverse video or underlined, depending on the type of terminal you are using.

The focus is created by ‘‘holding’’ one end at a certain position and moving the other end around. Pressing [hold-toggle] starts holding an end of the focus at the current cursor position; commands that normally move the cursor around (like [right], [down], [end-of-line], and [search-forward]) then only move the other end of the focus. If you now press [hold-toggle] again, the cursor moves back to the begin of the focus, and you are no longer holding it. If you now execute another motion command, the focus will disappear. As long as the focus is present, the text in it is *in danger*: if you start typing, the text in the focus is deleted, so that your typing in fact replaces the focus. To just delete the focus, press [backspace]. (If you accidentally hit the wrong key when you have made a focus, you can always [undo] the change, of course!)

Another way of creating a focus is the command [widen]. It creates successively larger foci: word, line, paragraph (delimited by blank lines), whole file. There are also commands [narrow-first] and [narrow-last] that move to the first or last subcomponent of the focus, respectively. For example, [narrow-first] moves from a paragraph to its first line, then to its first word, then to a cursor before its first character.

The focus is best appreciated for its capability to move or duplicate text. The command [cut] deletes the text in the focus but first saves a copy of it in a special buffer named the *Clipboard*. At any moment, the contents of the Clipboard can be inserted into the text you are editing by pressing [paste]. Normally this inserts the text right before the cursor, but if you have made a focus, that is first deleted (it is thus as if you typed the contents of the Clipboard yourself). You can insert as many copies of the Clipboard as you want. It is possible to transfer the contents of the focus to the Clipboard without also deleting it, with the command [copy]. Note the difference between [cut] and [backspace]: both delete the focus, but [cut] also copies it to the Clipboard, while [backspace] leaves the Clipboard unchanged.

If you want to inspect or edit the contents of the Clipboard, you can use [show-clipboard] to show it, but you will have to learn about windows first to know how to get rid of it again.

5.1.6. Searching and Replacing

5.1.6.1. Searching for Strings

When editing a large file, often you find that you know roughly the contents of a particular passage, but you do not know exactly where in the file it occurs. The editor can search for a particular string of characters with the commands [forward-search] and [reverse-search]. When you press either of these, a *prompt* appears on the bottom line of the screen, asking

you to type the string you want to search for. You can use most editing operations while typing the string, including [cut], [copy] and [paste], but excluding [undo]. When you have finished typing the string, press [return] and the search will start.

For [forward-search], the string is searched between the end of the current focus (or the cursor) and the end of the file; for [reverse-search] it is searched between the beginning of the focus or cursor and the beginning of the file, going backwards. The focus is moved to the first occurrence found; if the string is not found, the focus remains unchanged and the editor beeps at you.

Only strings that match exactly what you typed are found. For example, if you search for `exact` it will not find `Exact` nor `EXACT`, nor `ex` nor `ex-` on one line and `act` on the next line. On the other hand, a match is found even if it is part of a larger word, so `exactly` **is** found in this example.

If the string occurs several times in the file, you may want to repeat the search to see all its occurrences, or to find a particular occurrence that is interesting to you. You do not have to type the string again each time you start a new search; you can press [forward-search] [return] to repeat searching the same string. There is only one string that is being remembered by [forward-search] and [reverse-search], so you can also change directions while searching for the same string (this may be useful to make sure you have seen all occurrences in the file). You can also *dither* the search keys: [forward-search] [forward-search] is equivalent to [forward-search] [return].

When you have accidentally pressed a search command, you can back out by pressing [cancel].

5.1.6.2. Replacing Strings

Sometimes you find yourself changing all or many occurrences of a particular word or string to something else. The editor can help you with this, too: the [replace-string] command, which is akin to the [forward-search] command, first asks for a string and actually searches for it like [forward-search]. It then asks for another string, the *replacement*, and finally it asks if you want to replace the focus by the replacement.

You can type `yes` or `no` followed by [return] to answer the question (actually a single letter suffices, and pressing only [return] implies `yes`). If you typed `yes` the editor performs the replacement for you, and in all cases it starts a search for the next occurrence of the string, and asks again if you want to replace it (with the same replacement string), and so on until it beeps because no more occurrences of the string were found.

If you discover you do not want to do any more replacements even though there may be many more occurrences of the string, you can press [cancel] which aborts the [replace-string] command. If you are sure you want to replace all following occurrences of the string, you can answer `all` instead of `yes`, and then all following occurrences will be replaced with no questions asked; the focus moves to the last replacement.

5.1.6.3. Regular Expressions

For those who are used to other UNIX editors like *vi* or *ed*, there are variants of the above commands, [re-forward-search], [re-reverse-search] and [re-replace-string], that search for regular expressions instead of strings. Regular expressions are similar to those in *ed*; replacement strings are still only strings (no ‘&’ and ‘\1’; sorry). Using regular expressions

is not encouraged for casual users.

5.1.7. Some More Deletion Commands

For more experienced users, there are a few more commands to delete portions of the text. [delete-focus] deletes the character at the cursor rather than the character to the left of it like [backspace]. [delete-word-left] deletes from the cursor to the beginning of the previous word (that is, where [word-left] would bring you). Similarly, [delete-word-right] deletes from the cursor to the end of the next word. Finally, [delete-to-end-of-line] deletes from the cursor the the end of the line; when already at the end of the line, it joins the line with the next line.

Like [backspace], all commands mentioned in this section delete the focus if there is one, instead of performing their usual action.

5.1.8. The Right Margin

A right margin can be defined, which has the following effect: as soon as text is typed beyond the position of the right margin, the current line is split in two before the first word (delimited by white space) that does not fit on the line (and is not the first word of the line).

The command [right-margin-set] prompts for a number telling where the margin is; entering zero turns the right margin off. The default value is the last non-zero margin value given, or 72 initially.

Normally the right margin is not set, unless the **−m** command line option was given with a non-zero argument.

5.1.9. Windows and Files

Until now, we have assumed that we are editing a single file. In fact, *be* is capable of editing several files at once. Each file edited will be displayed in a separate portion of the screen; these portions are called *windows*. The *current window* is always the window with the cursor in it, and there are commands to move back and forth between windows, that is, changing the current window.

Several types of window exist, and the commands possible in a window depend on the window's type. Until now, we have only mentioned *regular* windows: windows on normal files. But there exist also *directory* windows, a *clipboard* window and a *key-bindings* window.

When *be* is invoked with several file arguments, it will immediately start off with each file in a separate window. However, it is also possible to create and delete windows dynamically. Regular windows are created using [new-file] or [visit-file]. Hitting [new-file] will cause *be* to prompt for a file name. A new window is then created, and, depending on whether the file exists and is readable, its contents are displayed. [Visit-file] is used to create a window on a file selected in a directory window. In fact, [visit-file] may be used to create several windows on several files thus selected.

A directory window is created by hitting [list-directory], or by hitting [visit-file] after selecting a directory. Within a directory window, file names can be selected by going to them ([down], [up] [search] or [goto]), and by hitting [hold]. The name will then appear highlighted. Deleting a file name in a directory window ([delete-focus] or [backspace]) effectively deletes the associated file. This can be undone ([undo]), but files thus removed

cannot be recovered after exiting *be*, so beware!

The *clipboard* window appears on hitting [show-clipboard]. The contents of the clipboard replace the focus on a [paste] command. Normal editing of the clipboard is possible, but it is not automatically saved when the editor is left.

Finally, the *key-bindings* window, is shown by [help], which is bound to ESC-ESC or ESC-?. The key-bindings cannot be modified by editing them.

Windows are removed by [delete-window] or by [delete-other-windows]. The former will delete the current window; the latter will delete all other windows. Before a normal window is actually deleted, the contents of its associated file are written.

Windows can be made current by hitting [next-window] or by ‘‘clicking them’’ (only possible when you have a mouse).

5.1.10. Backups

Every fifteen minutes or ˜1000 input characters (whichever comes first), *be* saves a backup copy of all modified files that have changed since the last backup was written. Normally you will hardly notice this (the backup is made in the background while you can continue typing in the foreground). The backup is removed whenever you save a file properly, so you leave little garbage behind.

How to recover lost changes from a backup?

1) Go to the directory (using *cd*) where you were when the editor crashed (this is necessary because the backup does not contain full path names).

2) Type

```
be -r
```

   This enters an interactive dialogue where you are presented with each backup file and the differences between it and its old version. It asks if it is OK to move the backup to the original, to which you may answer `yes` or `no`. If you do not trust the backup, answer `no`; the backup will remain where it is and you can first make a copy of the original and then call *be −r* again.

5.1.11. Record and Play Back

It is often the case that when editing, a certain sequence of commands must be done repeatedly. *Be* provides a way to do this simply. When the command [record-on] is given, *be* starts recording all they keys typed, both command and regular characters. While it is doing this, editing proceeds as normal. When the command [record-off] is given, *be* stops adding characters to the recording. Thereafter, whenever the command [play-back] is given, *be* ‘‘plays back’’ the recorded sequence of characters, just as if they had been typed in again. Whatever sort of editing changes were made the first time are repeated, but the starting point is wherever the cursor is, so you can make one change in one place, then move the cursor to a new place and do the same thing again, with one command.

5.1.12. Filtering

Be can filter input through an Amoeba command. Select some input with the focus and then press [filter-through-unix-command]. You are prompted for an Amoeba command, which can be something like *wc* or *spell*. The focus is used as input for this command. It disappears and is replaced by the output of the same command. This could be used, for example, to tabulate or typeset input. If you wish to filter the input through an Amoeba command, but do not want to loose it (as is probably the case if you are using *wc* or *spell*), then first copy the focus text to the clipboard, then show the clipboard, select the text to filter there, and send that through the filter command.

5.1.13. Preparing a Key Bindings File for BE

If *be* is called with the command line option **−b file**, a list of key bindings is read from this file. This option may occur more than once, and the bindings read from all files will be accumulated. Each line binds a key sequence to a command, except blank lines and lines beginning with # (which are treated as comments). The lines must have the following format:

```
command-name=binding
```

The command-name is an arbitrary string, but it must be one of the established command names or a unique prefix thereof. It may also be the name *ignore*, to indicate that a key sequence should not be treated as any command (this can be useful to remove bindings). No spaces are allowed in or around the name or the = sign. The binding contains the key sequence; it should be fairly short (say, less than 8 characters). It may contain the following escape sequences:

```
\b: backspace
\t: tab
\n: newline (=^J)
\f: formfeed (=^L)
\r: return
\e: ESC (ASCII 033 octal)
\ddd (ddd = up to 3 octal digits): the ASCII character ddd octal
^X (X = any upper case letter or one of ^ _ [ ] \ @: control-X
^?: DEL, ASCII 177 octal
\ followed by any other character except newline: that character
```

Note: Since all incoming characters are first stripped of their high bit, it is not useful to specify characters >= \200. Although the implementation uses null-terminated strings to represent bindings internally, it is safe to specify \000 as a character in a binding, since this is internally translated to \377.

The format of the key bindings file is the same as that of the help buffer, which can also be inspected as a source of examples and for an exhaustive list of default bindings. (The help command is by default bound to both ESC-ESC and ESC-?.) The help buffer can be dumped to a file by sucking its contents into the clipboard using [select-all] and [copy-focus-to-clipboard] and then pasting it into a new file.

5.1.14. The BE Command List

This file contains an alphabetical listing of all the available be commands. The bindings shown are default, and there may be additional or different bindings than the ones shown, depending upon the terminal type and setup.

backspace=^H

If the focus is non-empty, backspace deletes the text in the focus. Otherwise, it deletes the character before the cursor. If the cursor is at the begin of a line, the line is joined with the previous line.

begin-of-file=^[<

Moves the cursor to the first position of the file.

begin-of-line=^A

Moves the cursor to the beginning of the line. If a line has leading white space, it goes to the first non-blank character of the line, and a second execution of the command brings it to the real begin. (This command does not work on the Blit?)

copy=^[w

Copies the contents of the focus to the Clipboard. The focus must not be empty.

cut=^W

Moves the contents of the focus to the Clipboard. The focus is deleted from the file.

delete-focus=^D

If the focus is non-empty, deletes the text in the focus. Otherwise, deletes the character at the cursor. If the cursor is at the end of the line, joins it with the next line.

delete-other-windows=^X1

Deletes all windows except the current window, which is expanded to fill the whole screen. Documents in windows that are deleted are saved to their files if they have been modified.

delete-to-end-of-line=^K

Deletes from the current cursor position to the end of the line.

delete-window=^Xd

Deletes the current window. The document is saved to its file if it has been modified.

down-arrow=^N
down-arrow=^[OB
down-arrow=^[[B

Moves the cursor to the next line, in the same column if possible.

end-of-file=^[>

Moves the cursor to the end of the file.

end-of-line=^E

Moves the cursor to the end of the line. If it is already at the end of the line, it moves to the end of the next line.

filter-focus-through-unix-command=^[|

Prompts for a shell command. The focus is sent as standard input to this command.

The focus is then replaced with the output of this command.

forward-search=^[s

Prompts for a string which is searched between the end of the focus and the end of the file. The empty string repeats the previous search.

goto-cursor-position=^G
goto-cursor-position=^[g

On terminals which can ‘‘sense’’ the cursor (or mouse) position, moves the focus to the position of the cursor. If the cursor points to a different window, that window is selected but its focus remains unchanged.

goto-line=^[G

Prompts for a line number. Moves the cursor to the beginning of that line, or to the beginning of the last line of the file, if there are not that many lines. If the number is given with a beginning ‘+’ or ‘–’, it is interpreted relative to the current line number.

goto-tag=^X^T
goto-tag=^A@I

Prompts for a tag. If this tag is found in a tags file (see *ctags*(U)), the location in the documents containing that tag is brought into the window. (If the tag is in a file that is not yet being edited, a new window is created for it).

help=^[? help=^[^[

Prompts for a string which is used to select certain lines from the current list of key bindings. The empty string shows all key bindings.

hold-on=^Bi

‘‘Holds’’ one end of the focus. This means that all commands that move the focus will from now on only move the other end of the focus, so it may be grown or shrunk at will. Changes or undo will turn off the focus.

hold-off=^Bj

End the hold, so that the focus is fixed. If the next command is an editing command, the area in the focus is affected. Otherwise the focus is turned off.

hold-toggle=^@
hold-toggle=^[h

Change the state of the focus: if the focus was off, start it, and if it was on, stop it.

left-arrow=^B
left-arrow=^[OD
left-arrow=^[[D

Moves the cursor one position left. If at the begin of the line, move to the end of the previous line.

list-directory=^Xi

Creates a new window with a listing of the current directory in it.

literal-next-character=^\

The next character typed will be inserted in the buffer, even if it is a control character or otherwise special.

narrow-first=^[F

Narrow the focus to include only the first sub-element of the current focus. (For example, if the current document is a text document, the levels of elements are word, line, file, so if the focus were pointing to 3 lines, narrow-focus would reduce it to the first line only, and narrow-focus again would reduce it to the first word of the first line.)

narrow-last=^[L

The same as narrow-first, except the last element of the current focus is selected.

new-file=^X^V

Prompts for a file name, which is visited in a window. The file may be new or already exist. Shell variables, the ~/ and ~user/ abbreviations and shell metacharacters *, ? and [] may be used, as long as exactly one file is designated by the pattern.

newline=^J
newline=^M

Replaces the focus by a newline. The new line has the same indentation as the line before it.

newline-and-back-up=^O

Adds a newline after the cursor, but does not change the cursor position. If the cursor was at the end of a line, it adds a new empty line after; if the cursor was in the middle of a line, it splits the line.

next-window=^Xn

If there is more than one window on the screen, moves to the next window.

page-backward=^[v

Moves the cursor back approximately one window full.

page-forward=^V

Moves the cursor forward approximately one window full.

paste=^Y

Replaces the focus by the contents of the Clipboard, which must not be empty.

play-back=^Xe

Play back the previously saved recording. See record-on.

previous-window=^Xp

If there is more than one window on the screen, moves to the previous window.

quit-without-save=^C
quit-without-save=^X^C

Leaves the editor without saving any buffers. If there are any unsaved buffers, it asks whether you really want to quit.

re-forward-search=^[S

Prompts for a regular expression which is searched between the end of the focus and the end of the file. The empty string repeats the previous search.

re-replace-string=^[P

Like replace-string, except uses regular expressions.

re-reverse-search=^[R

Like re-forward-search but searches backward from the begin of the focus to the begin of the file.

record-on=^X(

Turns on recording. All the characters typed until the command record-off is received will be recorded. When the command play-back is given, the same sequence of characters will be issued, just as if they had been typed in again. By this mechanism, it is possible to repeat long sequences of identical commands very easily.

record-off=^X)

Turn off the recording. See record-on.

record-toggle=^ABD

If recording is off, turn it on. If on, turn it off. See record-on.

redo=^R

Immediately after executing undo one or more times, redo undoes the last undo. Repeated redo's undo the undo's before that. As soon as a change has been made (or even the focus has been moved), redo of an undo is no longer possible.

redraw-screen=^L

Clears the screen and redraws it, exactly as it should be.

replace-string=^[p

Prompts for a search string, then for a string to replace it with. For each occurrence of the search string found, it asks if you want to replace it with the replace string, or you can indicate 'all' to replace all occurrences of the search string with the replace string (with no further prompting).

reverse-search=^[r

Like forward-search but searches backward from the begin of the focus to the begin of the file.

right-arrow=^F
right-arrow=^[OC
right-arrow=^[[C

Moves the cursor one position to the right. If at the end of a line, moves to the begin of the next line.

right-margin-set=^Xm

Set the value of the right margin. If 0, the right margin is turned off.

right-margin-toggle=^ABG

If the right margin is on, turns it off. If off, turns it on.

save-all-files=^X^M

Saves all files that have been modified, without closing their windows.

save-files-and-quit=^X^F

Saves all files that have been modified and quits from the editor. If not all files could be saved, it does not quit; you can quit with 'quit-without-save' then.

scroll-backward=^[v

Scroll the screen backward by one screen full.

scroll-forward=^[[70;1H

Scroll the screen forward by one screen full.

select-all=^[a

Sets the focus to the entire file (useful for copying entire files via the Clipboard).

show-clipboard=^Xc

Shows the Clipboard in a window. The Clipboard can be edited, but is not saved between sessions.

show-index=^Xi

Shows the directory index of the current document. In a directory index, editing commands have a different meaning: typing a character searches the first file whose name begins with that character, and the focus can only be moved up and down. Lines can be placed in and removed from a ‘selection’, which is shown in reverse video. The command to do this is ‘hold’, which operates as a toggle.

show-workspaces=^XI

Shows the index of indices, in other words the list of directories. It only shows directories from which at least one file has been visited. This index behaves in a way similar to a directory index.

suspend-editor=^_

Sends the editor (and its process group) a SIGSTOP signal, which suspends it temporarily (if your shell knows about ‘‘job control’’). The terminal is properly reset. When an *fg* command brings the editor process back to the foreground, it continues its session. No files are saved by this command.

swap-two-previous-characters=^T

Interchange the two characters before the cursor.

tabulate=^I

Inserts a tab in the buffer.

undo=^Z

Undoes the last command or sequence of similar commands. Typing characters, backspacing and short-range focus moves are treated as a unit.

up-arrow=^P
up-arrow=^[OA
up-arrow=^[[A

Moves the cursor to the previous line, in the same column of possible.

visit-file=^Xv

This command only works in a directory index or in the index of directories. It opens a window on all files in the selection, or on the file at the cursor if the selection is empty.

widen-focus=^[W

Widen the focus to include the next larger element: from nothing or a few characters to

a word, from word(s) to a line, from line(s) to a paragraph, from paragraph(s) to the whole file.

word-left=^Bf

Move the cursor one word to the left. If on the end of a line, moves the cursor to the first word on the next line. Words are anything that is surrounded by spaces (including numbers, punctuation, and so forth).

word-right=^Bg

Like word-left, but to the right instead. If at the beginning of a line, moves the cursor to the last word on the previous line.

5.2. ed

The Minix version of the editor *ed* has been ported to Amoeba. The manual page (see *ed*(U)) describes how this editor works.

5.3. Elvis Reference Manual

By Steve Kirkendall (E-Mail: kirkenda@cs.pdx.edu)

1. INTRODUCTION
 1.1 Compiling
 1.2 Overview of Elvis

2. VISUAL MODE COMMANDS
 2.1 Input Mode
 2.2 Arrow keys in Input Mode
 2.3 Digraphs
 2.4 Abbreviations
 2.5 Auto-Indent

3. COLON MODE COMMANDS
 3.1 Line Specifiers
 3.2 Text Entry Commands
 3.3 Cut & Paste Commands
 3.4 Display Text Commands
 3.5 Global Operations Commands
 3.6 Line Editing Commands
 3.7 Undo Command
 3.8 Configuration & Status Commands
 3.9 Multiple File Commands
 3.10 Switching Files
 3.11 Working with a Compiler
 3.12 Exit Commands
 3.13 File I/O Commands
 3.14 Directory Commands
 3.15 Debugging Commands

4. REGULAR EXPRESSIONS
 4.1 Syntax
 4.2 Options
 4.3 Substitutions
 4.4 Examples

5. OPTIONS
 5.1 AutoIndent
 5.2 AutoPrint
 5.3 AutoWrite
 5.4 CC
 5.5 CharAttr
 5.6 COlumns

5.7 DIGraph
5.8 DIRectory
5.9 EDcompatible
5.10 ErrorBells
5.11 ExRefresh
5.12 FlipCase
5.13 HideFormat
5.14 IgnoreCase
5.15 InputMode
5.16 KeyTime
5.17 KeywordPrg
5.18 LiNes
5.19 LIst
5.20 MAgic
5.21 MaKe
5.22 ModeLine
5.23 PAragraphs
5.24 ReadOnly
5.25 REport
5.26 SCroll
5.27 SEctions
5.28 SHell
5.29 ShiftWidth
5.30 ShowMatch
5.31 ShowMoDe
5.32 SideScroll
5.33 SYnc
5.34 TabStop
5.35 TErm
5.36 VBell
5.37 WArn
5.38 WrapMargin
5.39 WrapScan

6. CUT BUFFERS
6.1 Filling
6.2 Pasting from a Cut Buffer
6.3 Macros
6.4 The Effect of Switching Files

7. DIFFERENCES BETWEEN ELVIS & BSD VI/EX
7.1 Extensions
7.2 Omissions

8. INTERNAL
8.1 The temporary file
8.2 Implementation of Editing

8.3 Marks and the Cursor
8.4 Colon Command Interpretation
8.5 Screen Control
8.6 Portability

9. CFLAGS

10. TERMCAP
10.1 Required numeric fields
10.2 Required string fields
10.3 Boolean fields
10.4 Optional string fields
10.5 Optional strings received from the keyboard
10.6 Optional fields that describe character attributes
10.7 Optional fields that affect the cursor's shape
10.8 An example

11. ENVIRONMENT VARIABLES
11.1 TERM, TERMCAP
11.2 TMP, TEMP
11.3 EXINIT
11.4 SHELL, COMSPEC
11.5 HOME

12. VERSIONS
12.1 BSD UNIX
12.2 System-V UNIX
12.3 SCO Xenix
12.4 Minix
12.5 Coherent
12.6 MS-DOS
12.7 Atari TOS
12.8 OS9/6

5.3.1. INTRODUCTION

Elvis is a clone of *vi/ex*, the standard UNIX editor. *Elvis* supports nearly all of the *vi/ex* commands, in both *visual mode* and *colon mode*.

Like *vi/ex*, *elvis* stores most of the text in a temporary file, instead of RAM. This allows it to edit files that are too large to fit in a single process' data space.

Elvis runs under Amoeba, BSD UNIX, AT&T SysV UNIX, Minix, MS-DOS, Atari TOS, Coherent, and OS9/68000. The next version is expected to add OS/2, VMS, AmigaDos, and MacOS. Contact the author before you start porting it to some other OS, because somebody else may have already done it.

Elvis is freely redistributable, in either source form or executable form. There are no restrictions on how you may use it.

5.3.1.1. Compiling

See the ''Versions'' section of this manual for instructions on how to compile *elvis*.

If you want to port *elvis* to another OS or compiler, then you should read the ''Portability'' part of the ''Internal'' section.

5.3.1.2. Overview of Elvis

The user interface of *elvis/vi/ex* is weird. There are two major command modes in *elvis*, and a few text input modes as well. Each command mode has a command which allows you to switch to the other mode.

You will probably use the *visual mode* most of the time. This is the mode that *elvis* normally starts up in.

In visual command mode, the entire screen is filled with lines of text from your file. Each keystroke is interpreted as part of a visual command. If you start typing text, it will **not** be inserted, it will be treated as part of a command. To insert text, you must first give an ''insert text'' command. This may take some getting used to. (An alternative exists. Lookup the ''inputmode'' option.)

The *colon mode* is quite different. *Elvis* displays a '':'' character on the bottom line of the screen, as a prompt. You are then expected to type in a command line and hit the <Return> key. The set of commands recognized in the colon mode is different from visual mode's.

The rest of this manual describes the commands available under each command mode followed by a description of the many options available. Thereafter comes a lot of information about the internal workings of *elvis* and other details which are mainly relevant to those compiling and installing *elvis*.

5.3.2. VISUAL MODE COMMANDS

Most visual mode commands are one keystroke long. The following table lists the operation performed by each keystroke, and also denotes any options or arguments that it accepts. Notes at the end of the table describe the notation used in this table.

In addition to the keys listed here, your keyboard's ''arrow'' keys will be interpreted as the appropriate cursor movement commands. The same goes for <PgUp> and <PgDn>, if your keyboard has them. There is a colon mode command (to be described later) which will allow

you to define other keys, such as function keys.

A tip: visual command mode looks a lot like text input mode. If you forget which mode you are in, just hit the <Esc> key. If *elvis* beeps, then you are in visual command mode. If *elvis* does not beep, then you were in input mode, but by hitting <Esc> you will have switched to visual command mode. So, one way or another, after <Esc> *elvis* will be ready for a command.

command			**description**	**type**
	^A		—	
	^B		Move toward the top of the file by 1 screen full	
	^C		—	
count	^D		scroll down <count> lines (default 1/2 screen)	
count	^E		scroll up <count> lines	
	^F		move toward the bottom of the file by 1 screen full	
	^G		show file status, and the current line #	
count	^H		move left, like h	MOVE
	^I		—	
count	^J		move down	MOVE
	^K		—	
	^L		redraw the screen	
count	^M		move to the front of the next line	MOVE
count	^N		move down	MOVE
	^O		—	
count	^P		move up	MOVE
	^Q		—	
	^R		redraw the screen	
	^S		—	
	^T		—	
count	^U		scroll up <count> lines (default 1/2 screen)	
	^V		—	
	^W		—	
	^X		—	
count	^Y		scroll down <count> lines	
	^Z		—	
	ESC		—	
	^\		—	
	^]		if the cursor is on a tag name, go to that tag	
	^^		switch to the previous file, like ‘‘:e #’’	
	^_		—	
count	SPC		move right,like ‘‘l’’	MOVE
	!	mv	run the selected lines through an external filter program	
	"	key	select which cut buffer to use next	
count	#	+	increment a number	EDIT
	$		move to the rear of the current line	MOVE
	%		move to the matching (){}[] character	MOVE
count	&		repeat the previous ‘‘:s//’’ command here	EDIT
	’	key	move to a marked line	MOVE

count	(		move backward <count> sentences	MOVE
count	)		move forward <count> sentences	MOVE
	*		go to the next error in the errlist	
count	+		move to the front of the next line	MOVE
count	,		repeat the previous [fFtT] but in the other direction	MOVE
count	−		move to the front of the preceding line	MOVE
count	.		repeat the previous ‘‘edit’’ command	
	/text		search forward for a given regular expression	MOVE
	0		if not part of count, move to first char of this line	MOVE
	1		part of count	
	2		part of count	
	3		part of count	
	4		part of count	
	5		part of count	
	6		part of count	
	7		part of count	
	8		part of count	
	9		part of count	
	:	text	run single *ex* cmd	
count	;		repeat the previous [fFtT] cmd	MOVE
	<	mv	shift text left	EDIT
	=		—	
	>	mv	shift text right	EDIT
	?	text	search backward for a given regular expression	MOVE
	@	key	execute the contents of a cut-buffer as *vi* commands	
count	A	inp	append at end of the line	EDIT
count	B		move back Word	MOVE
	C	inp	change text from the cursor through the end of the line	EDIT
	D		delete text from the cursor through the end of the line	EDIT
count	E		move end of Word	MOVE
count	F	key	move leftward to a given character	MOVE
count	G		move to line #<count> (default is the bottom line)	MOVE
count	H		move to home row (the line at the top of the screen)	
count	I	inp	insert at the front of the line (after indents)	EDIT
count	J		join lines, to form one big line	EDIT
	K		look up keyword	
count	L		move to last row (the line at the bottom of the screen)	
	M		move to middle row	
	N		repeat previous search, but in the opposite direction	MOVE
count	O	inp	open up a new line above the current line	EDIT
	P		paste text before the cursor	
	Q		quit to *ex* mode	
	R	inp	overtype	EDIT
count	S	inp	change lines, like <count>cc	
count	T	key	move leftward *almost* to a given character	MOVE
	U		Undo all recent changes to the current line	
	V		—	

count	W		move forward <count> Words	MOVE
count	X		delete the character(s) to the left of the cursor	EDIT
count	Y		yank text line(s) (copy them into a cut buffer)	
	ZZ		save the file & exit	
	[[		move back 1 section	MOVE
	\		—	
	]]		move forward 1 section	MOVE
	^		move to the front of the current line (after indent)	MOVE
	_		—	
	‘		key move to a marked character	MOVE
count	a	inp	insert text after the cursor	EDIT
count	b		move back <count> words	MOVE
	c	mv	change text	EDIT
	d	mv	delete text	EDIT
count	e		move forward to the end of the current word	MOVE
count	f	key	move rightward to a given character	MOVE
	g		—	
count	h		move left	MOVE
count	i	inp	insert text at the cursor	EDIT
count	j		move down	MOVE
count	k		move up	MOVE
count	l		move right	MOVE
	m	key	mark a line or character	
	n		repeat the previous search	MOVE
count	o	inp	open a new line below the current line	EDIT
	p		paste text after the cursor	
	q		—	
count	r	key	replace <count> chars by a given character	EDIT
count	s	inp	replace <count> chars with text from the user	EDIT
count	t	key	move rightward *almost* to a given character	MOVE
	u		undo the previous edit command	
	v		—	
count	w		move forward <count> words	MOVE
count	x		delete the character that the cursor’s on	EDIT
	y	mv	yank text (copy it into a cut buffer)	
	z	key	scroll current line to the screen’s +=top −=bottom .=middle	
count	{		move back <count> paragraphs	MOVE
count	|		move to column <count> (the leftmost column is 1)	
count	}		move forward <count> paragraphs	MOVE
count	~		switch a character between uppercase & lowercase	EDIT
	DEL		—	

count

Many commands may be preceded by a count. This is a sequence of digits representing a decimal number. For most commands that use a count, the command is repeated <count> times. The count is always optional, and usually defaults to 1.

key

Some commands require two keystrokes. The first key always determines which command is to be executed. The second key is used as a parameter to the command.

mv
: Six commands (! < > c d y) operate on text between the cursor and some other position. To specify that other position, you are expected to follow the command keystroke with a movement command. Also, you may have the command operate on the whole line that the cursor is on by typing the command key a second time.

inp
: Many commands allow the user to interactively enter text.

EDIT
: These commands affect text, and may be repeated by the ‘‘.’’ command.

MOVE
: These commands move the cursor, and may be used to specify the extent of a member of the ‘‘mv’’ class of commands.

5.3.2.1. Input Mode

You cannot type text into your file directly from visual command mode. Instead, you must first give a command which will put you into input mode. The commands to do this are A/C/I/O/R/S/a/i/o/s.

The S/s/C/c commands temporarily place a $ at the end of the text that they are going to change.

In input mode, all keystrokes are inserted into the text at the cursor’s position, except for the following:

^A	insert a copy of the last input text
^D	delete one indent character
^H	(backspace) erase the character before the cursor
^L	redraw the screen
^M	(carriage return) insert a newline (^J, linefeed)
^P	insert the contents of the cut buffer
^R	redraw the screen, like ^L
^T	insert an indent character
^U	backspace to the beginning of the line
^V	insert the following keystroke, even if special
^W	backspace to the beginning of the current word
^Z^Z	write the file & exit *elvis*
^[	(ESCape) exit from input mode, back to command mode

Also, on some systems, ^S may stop output, ^Q may restart output, and ^C may interrupt execution. ^@ (the NUL character) cannot be inserted.

The R visual command puts you in overtype mode, which is a slightly different form of input mode. In overtype mode, each time you insert a character, one of the old characters is deleted from the file.

5.3.2.2. Arrow keys in Input Mode

The arrow keys can be used to move the cursor in input mode. (This is an extension; the real *vi* does not support arrow keys in input mode.) The <PgUp>, <PgDn>, <Home>, and <End> keys work in input mode, too. The <Delete> key deletes a single character in input mode.

The best thing about allowing arrow keys to work in input mode is that as long as you are in input mode, *elvis* seems to have a fairly ordinary user interface. With most other text editors, you are always in either insert mode or replace mode, and you can use the arrow keys at any time to move the cursor. Now, *elvis* can act like that, too. In fact, with the new ‘‘inputmode’’ option and the ‘‘CTRL−Z CTRL−Z’’ input command, you may never have to go into visual command mode for simple edit sessions.

5.3.2.3. Digraphs

Elvis supports digraphs as a way to enter non-ASCII characters. A digraph is a character which is composed of two other characters. For example, an apostrophe and the letter i could be defined as a digraph which is to be stored and displayed as an accented i.

There is no single standard for extended ASCII character sets. *Elvis* can be compiled to fill the digraph with values appropriate for either the IBM PC character set, or the LATIN−1 character set used by X windows, or neither. (See the discussions of `−DCS_IBMPC` and `−DCS_LATIN1` in the CFLAGS section of this manual.) You can view or edit the digraph table via the ‘‘:digraph’’ colon command.

Digraphs wil not be recognized until you have entered ‘‘:set digraph’’.

To actually use a digraph type the first character, then hit <Backspace>, and then type the second character. *Elvis* will then substitute the non-ASCII character in their place.

5.3.2.4. Abbreviations

Elvis can expand abbreviations for you. You define an abbreviation with the `:abbr` command, and then whenever you type in the abbreviated form while in input mode, *elvis* will immediately use the long form.

Elvis does not perform the substitution until you type a non-alphanumeric character to mark the end of the word. If you type a CTRL−V before that non-alphanumeric character, then *elvis* will not perform the substitution.

5.3.2.5. Auto-Indent

With the ‘‘:set autoindent’’ option turned on, *elvis* will automatically insert leading white space at the beginning of each new line that you type in. The leading white space is copied from the preceding line.

To add more leading white space, type CTRL−T. To remove some white space, type CTRL−D.

Elvis’ autoindent mode is not 100% compatible with *vi*’s. In *elvis*, 0ˆD and ˆˆD do not work, ˆU can wipe out all indentation, and sometimes *elvis* will use a different amount of indentation than *vi* would.

5.3.3. COLON MODE COMMANDS

lines	command	arguments
[line]	**a**ppends	
	args	[files]
	cc	[files]
	cd	[directory]
[line][,line]	**c**hange	
	chdir	[directory]
[line][,line]	**co**py	line
[line][,line]	**d**elete	["x]
	digraph[!]	[XX [Y]]
	edit[!]	[file]
	errlist[!]	[errlist]
	ex[!]	[file]
	file	[file]
[line][,line]	**g**lobal	/regexp/ command
[line]	**i**nsert	
[line][,line]	**j**oin	
[line][,line]	**l**ist	
	make	[target]
	map[!]	key mapped_to
[line]	mar**k**	x
	mkexrc	
[line][,line]	**m**ove	line
	next[!]	[files]
	Next[!]	
[line][,line]	**nu**mber	
	previous[!]	
[line][,line]	**p**rint	
[line]	**pu**t	["x]
	quit[!]	
[line]	**r**ead	file
	rewind[!]	
	set	[options]
	source	file
[line][,line]	**s**ubstitute	/regexp/replacement/[p][g][c]
	tag[!]	tagname
[line][,line]	**t**o	line
	undo	
	unmap[!]	ey
	version	
[line][,line]	**v**global	regexp/ command
	visual	
	wq	
[line][,line]	**w**rite[!]	[[>>]file]
	xit[!]	

[line][,line]	**y**ank	["x]
[line][,line]	**!**	command
[line][,line]	<	
[line][,line]	=	
[line][,line]	>	
[line][,line]	**&**	
	@	"x

To use colon mode commands, you must switch from visual command mode to colon command mode. The visual mode commands to do this are ‘‘:’’ for a single colon command, or ‘‘Q’’ for many colon mode commands.

5.3.3.1. Line Specifiers

Line specifiers are always optional. The first line specifier of most commands usually defaults to the current line. The second line specifier usually defaults to be the same as the first line specifier. Exceptions are :write, :global, and :vglobal, which act on all lines of the file by default, and :!, which acts on no lines by default.

Line specifiers consist of an absolute part and a relative part. The absolute part of a line specifier may be either an explicit line number, a mark, a dot to denote the current line, a dollar sign to denote the last line of the file, or a forward or backward search.

An explicit line number is simply a decimal number, expressed as a string of digits.

A mark is typed in as an apostrophe followed by a letter. Marks must be set before they can be used. You can set a mark in visual command mode by typing ‘‘m’’ and a letter, or you can set it in colon command mode via the ‘‘mark’’ command.

A forward search is typed in as a regular expression surrounded by slash characters; searching begins at the default line. A backward search is typed in as a regular expression surrounded by question marks; searching begins at the line before the default line.

If you omit the absolute part, then the default line is used.

The relative part of a line specifier is typed as a ‘‘+’’ or ‘‘−’’ character followed by a decimal number. The number is added to or subtracted from the absolute part of the line specifier to produce the final line number.

As a special case, the % character may be used to specify all lines of the file. It is roughly equivalent to saying 1,$. This can be a handy shortcut.

Some examples:

:p	print the current line
:37p	print line 37
:’gp	print the line which contains mark g
:/foo/p	print the next line that contains ‘‘foo’’
:$p	print the last line of the file
:20,30p	print lines 20 through 30
:1,$p	print all lines of the file
:%p	print all lines of the file
:/foo/−2,+4p	print 5 lines around the next ‘‘foo’’

5.3.3.2. Text Entry Commands

```
[line] append
[line][,line] change ["x]
[line] insert
```

The **a**ppend command inserts text after the specified line.

The **i**nsert command inserts text before the specified line.

The **c**hange command copies the range of lines into a cut buffer, deletes them, and inserts new text where the old text used to be.

For all of these commands, you indicate the end of the text you are inserting by hitting ^D or by entering a line which contains only a period.

5.3.3.3. Cut & Paste Commands

```
[line][,line] delete ["x]
[line][,line] yank ["x]
[line] put ["x]
[line][,line] copy line
[line][,line] to line
[line][,line] move line
```

The **d**elete command copies the specified range of lines into a cut buffer, and then deletes them.

The **y**ank command copies the specified range of lines into a cut buffer, but does *not* delete them.

The **pu**t command inserts text from a cut buffer after the specified line.

The **co**py and **t**o commands yank the specified range of lines and then immediately paste them after some other line.

The **m**ove command deletes the specified range of lines and then immediately pastes them after some other line. If the destination line comes after the deleted text, then it will be adjusted automatically to account for the deleted lines.

5.3.3.4. Display Text Commands

```
[line][,line] print
[line][,line] list
[line][,line] number
```

The **p**rint command displays the specified range of lines.

The **nu**mber command displays the lines, with line numbers.

The **l**ist command also displays them, but it is careful to make control characters visible.

5.3.3.5. Global Operations Commands

```
[line][,line] global /regexp/ command
[line][,line] vglobal /regexp/ command
```

The **g**lobal command searches through the lines of the specified range (or through the whole file if no range is specified) for lines that contain a given regular expression. It then moves the cursor to each of these lines and runs some other command on them.

The **v**global command is similar, but it searches for lines that **do not** contain the regular expression.

5.3.3.6. Line Editing Commands

```
[line][,line] join
[line][,line] ! program
[line][,line] <
[line][,line] >
[line][,line] substitute /regexp/replacement/[p][g][c]
[line][,line] &
```

The **j**oin command catenates all lines in the specified range together to form one big line. If only a single line is specified, then the following line is catenated onto it.

The **!** command runs an external filter program, and feeds the specified range of lines to its *stdin*. The lines are then replaced by the output of the filter. A typical example would be ‘‘:’a,’z!sort’’ to sort the lines ’a,’z.

The < and > commands shift the specified range of lines left or right, normally by the width of 1 tab character. The ‘‘shiftwidth’’ option determines the shifting amount.

The **s**ubstitute command finds the regular expression in each line, and replaces it with the replacement text. The ‘‘p’’ option causes the altered lines to be printed. The ‘‘g’’ option permits all instances of the regular expression to be found & replaced. (Without ‘‘g’’, only the first occurrence in each line is replaced.) The ‘‘c’’ option asks for confirmation before each substitution.

The **&** command repeats the previous substitution command. Actually, ‘‘&’’ is equivalent to ‘‘s//~/’’ with the same options as last time. It searches for the last regular expression that you specified for any purpose, and replaces it with the the same text that was used in the previous substitution.

5.3.3.7. Undo Command

```
undo
```

The **u**ndo command restores the file to the state it was in before your most recent command which changed text.

5.3.3.8. Configuration & Status Commands

```
map[!] [key mapped_to]
unmap[!] key
abbr [word expanded_form_of_word]
unabbr word
digraph[!] [XX [Y]]
set [options]
mkexrc
[line] mark "x
visual
version
[line][,line] =
file [file]
source file
@ "x
```

The **ma**p command allows you to configure *elvis* to recognize your function keys, and treat them as though they transmitted some other sequence of characters. Normally this mapping is done only when in the visual command mode, but with the [!] present it will map keys under all contexts. When this command is given with no arguments, it prints a table showing all mappings currently in effect. When called with two arguments, the first is the sequence that your function key really sends, and the second is the sequence that you want *elvis* to treat it as having sent.

The **unm**ap command removes key definitions that were made via the map command.

The **ab**br command is used to define/list a table of abbreviations. The table contains both the abbreviated form and the fully spelled-out form. When you are in visual input mode, and you type in the abbreviated form, *elvis* will replace the abbreviated form with the fully spelled-out form. When this command is called without arguments, it lists the table; with two or more arguments, the first argument is taken as the abbreviated form, and the rest of the command line is the fully spelled-out form.

The **una**bbr command deletes entries from the abbr table.

The **di**graph command allows you to display the set of digraphs that *elvis* is using, or add/remove a digraph. To list the set of digraphs, use the digraph command with no arguments. To add a digraph, you should give the digraph command two arguments. The first argument is the two ASCII characters that are to be combined; the second is the non-ASCII character that they represent. The non-ASCII character's most significant bit is automatically set by the digraph command, unless you append a ''!'' to the command name.

Removal of a digraph is similar to adding a digraph, except that you should leave off the second argument.

The **se**t command allows you examine or set various options. With no arguments, it displays the values of options that have been changed. With the single argument ''all'' it displays the values of all options, regardless of whether they have been explicitly set or not. Otherwise, the arguments are treated as options to be set.

The **mk**exrc command saves the current configuration to a file called *.exrc* in the current directory.

The mar**k** command defines a named mark to refer to a specific place in the file. This mark may be used later to specify lines for other commands.

The **vi**sual command puts the editor into visual mode. Instead of emulating *ex*, *elvis* will start emulating *vi*.

The **ve**rsion command tells you that what version of *elvis* this is.

The = command tells you what line you specified, or, if you specified a range of lines, it will tell you both endpoints and the number of lines included in the range.

The **f**ile command tells you the name of the file, whether it has been modified, the number of lines in the file, and the current line number. You can also use it to change the name of the current file.

The **so**urce command reads a sequence of colon mode commands from a file, and interprets them.

The **@** command executes the contents of a cut-buffer as *ex* commands.

5.3.3.9. Multiple File Commands

```
args [files]
next[!] [files]
Next[!]
previous[!]
rewind[!]
```

When you invoke *elvis* from your shell's command line, any file names that you give to *elvis* as arguments are stored in the args list. The **ar**gs command will display this list, or define a new one.

The **n**ext command switches from the current file to the next one in the args list. You may specify a new args list here, too.

The **N**ext and **pre**vious commands (they are really aliases for the same command) switch from the current file to the preceding file in the args list.

The **rew**ind command switches from the current file to the first file in the args list.

5.3.3.10. Switching Files

```
edit[!] [file]
tag[!] tagname
```

The **e**dit command allows to switch from the current file to some other file. This has nothing to do with the args list, by the way.

The **ta**g command looks up a given tagname in a file called ''tags''. This tells it which file the tag is in, and how to find it in that file. *Elvis* then switches to the tag's file and finds the tag.

5.3.3.11. Working with a Compiler

```
cc [files]
make [target]
errlist[!] [errlist]
```

The **cc** and **mak**e commands execute your compiler or *make* utility and redirect any error messages into a file called *errlist*. By default, *cc* is run on the current file. (You should write it out before running *cc*.) The contents of the *errlist* file are then scanned for error messages. If an error message is found, then the cursor is moved to the line where the error was detected, and the description of the error is displayed on the status line.

After you have fixed one error, the **er**rlist command will move the cursor to the next error. In visual command mode, hitting ‘‘*’’ will do this, too.

You can also create an *errlist* file from outside *elvis* and use

```
elvis -m
```

to start *elvis* and have the cursor moved to the first error. Note that you do not need to supply a file name with

```
elvis -m
```

because the error messages always say which source file an error is in.

Note: When you use *errlist* repeatedly to fix several errors in a single file, it will attempt to adjust the reported line numbers to allow for lines that you have inserted or deleted. These adjustments are made with the assumption that you will work though the file from the beginning to the end.

5.3.3.12. Exit Commands

```
quit[!]
wq
xit
```

The **q**uit command exits from the editor without saving your file.

The **wq** command writes your file out, then then exits.

The **x**it command is similar to the **wq** command, except that **x**it will not bother to write your file if you have not modified it.

5.3.3.13. File I/O Commands

```
[line] read file
[line][,line] write[!] [[>>]file]
```

The **r**ead command gets text from another file and inserts it after the specified line. It can also read the output of a program; simply precede the program name by a ‘‘!’’ and use it in place of the file name.

The **w**rite command writes the whole file, or just part of it, to some other file. The ‘‘!’’, if present, will permit the lines to be written even if you have set the readonly option. If you

precede the file name by >> then the lines will be appended to the file.

5.3.3.14. Directory Commands

```
cd [directory]
chdir [directory]
shell
```

The **cd** and **chd**ir commands (really two names for one command) switch the current working directory.

The **sh**ell command starts an interactive shell.

5.3.3.15. Debugging Commands

```
[line][,line] debug[!]
validate[!]
```

These commands are only available if you compile *elvis* with the `-DDEBUG` flag.

The de**b**ug command lists statistics for the blocks which contain the specified range of lines. If the ‘‘!’’ is present, then the contents of those blocks is displayed, too.

The **va**lidate command checks certain variables for internal consistency. Normally it does not output anything unless it detects a problem. With the ‘‘!’’, though, it will always produce *some* output.

5.3.4. REGULAR EXPRESSIONS

Elvis uses regular expressions for searching and substitutions.

5.3.4.1. Syntax

Elvis’ regexp package treats the following one or two character strings (called meta-characters) in special ways:

\(\)	Used to delimit subexpressions
^	Matches the beginning of a line
$	Matches the end of a line
\<	Matches the beginning of a word
\>	Matches the end of a word
[ ]	Matches any single character inside the brackets
*	The preceding may be repeated 0 or more times
\+	The preceding may be repeated 1 or more times
\?	The preceding is optional

Anything else is treated as a normal character which must match exactly. The special strings may all be preceded by a backslash to force them to be treated normally.

5.3.4.2. Options

Elvis has two options which affect the way regular expressions are used. These options may be examined or set via the :set command.

The first option is called *[no]magic*. This is a boolean option, and it is *magic* (TRUE) by default. While in magic mode, all of the meta-characters behave as described above. In *nomagic* mode, only ^ and $ retain their special meaning.

The second option is called *[no]ignorecase*. This is a boolean option, and it is *noignorecase* (FALSE) by default. While in *ignorecase* mode, the searching mechanism will not distinguish between an uppercase letter and its lowercase form. In *noignorecase* mode, uppercase and lowercase are treated as being different.

Also, the *[no]wrapscan* option affects searches.

5.3.4.3. Substitutions

The :s command has at least two arguments: a regular expression, and a substitution string. The text that matched the regular expression is replaced by text which is derived from the substitution string.

Most characters in the substitution string are copied into the text literally but a few have special meaning:

&	Insert a copy of the original text
~	Insert a copy of the previous replacement text
\1	Insert a copy of that portion of the original text which matched the first set of \(\) parentheses.
\2 − \9	Does the same for the second (etc.) pair of \(\).
\U	Convert all chars of any later &, ~, or \# to uppercase
\L	Convert all chars of any later &, ~, or \# to lowercase
\E	End the effect of \U or \L
\u	Convert the first char of the next &, ~ or \# to uppercase
\l	Convert the first char of the next &, ~ or \# to lowercase

These may be preceded by a backslash to force them to be treated normally. If *nomagic* mode is in effect, then & and ~ will be treated normally, and you must write them as \& and \~ for them to have special meaning.

5.3.4.4. Examples

This example changes every occurrence of ‘‘utilize’’ to ‘‘use’’:

```
:%s/utilize/use/g
```

The next example deletes all white space that occurs at the end of a line anywhere in the file. (The brackets contain a single space and a single tab.):

```
:%s/[ ]\+$//
```

The next example converts the current line to uppercase:

```
:s/.*/\U&/
```

This example underlines each letter in the current line, by changing it into an ‘‘underscore

backspace letter'' sequence. (The ^H is entered as ''CTRL−V backspace''.):

```
:s/[a-zA-Z]/_^H&/g
```

This example locates the last colon in a line, and swaps the text before the colon with the text after the colon. The first \(\) pair is used to delineate the stuff before the colon, and the second pair delineates the stuff after. In the substitution text, \1 and \2 are given in reverse order, to perform the swap:

```
:s/\(.*\):\(.*\)/\2:\1/
```

5.3.5. OPTIONS

Options may be set or examined via the colon command ''set''. The values of options will affect the operation of later commands.

For convenience, options have both a long descriptive name and a short name which is easy to type. You may use either name interchangeably.

long name	short name	type	default	meaning
autoindent	ai	Bool	noai	auto-indent during input
autoprint	ap	Bool	ap in *ex*,	print the current line
autowrite	aw	Bool	noaw	auto-write when switching files
charattr	ca	Bool	noca	interpret \fX sequences?
cc	cc	Str	cc="cc −c"	name of the C compiler
columns	co	Num	co=80	width of the screen
digraph	dig	Bool	nodig	recognize digraphs?
directory	dir	Str	dir="/usr/tmp"	where tmp files are kept
edcompatible	ed	Bool	noed	remember ":s//" options
errorbells	eb	Bool	eb	ring bell on error
exrefresh	er	Bool	er	write lines individually in *ex*
flipcase	fc	Str	fc=""	non-ASCII chars flipped by ~
hideformat	hf	Bool	hf	hide text formatter commands
ignorecase	ic	Bool	noic	upper/lowercase match in search
inputmode	im	Bool	noim	start vi in insert mode?
keytime	kt	Num	kt=2	timeout for mapped key entry
keywordprg	kp	Str	kp="ref"	full path name of shift−K prog
lines	ln	Num	ln=25	number of lines on the screen
list	li	Bool	noli	display lines in ''list'' mode
magic	ma	Bool	ma	use regular expression in search
make	mk	Str	mk="make"	name of the ''make'' program
modeline	ml	Bool	noml	are modelines processed?
paragraphs	pa	Str	pa="PPppIPLPQP"	names of ''paragraph'' nroff cmd
readonly	ro	Bool	noro	prevent overwriting of orig file
report	re	Num	re=5	report when 5 or more changes
scroll	sc	Num	sc=12	scroll amount for ^U and ^D
sections	se	Str	se="NHSHSSSEse"	names of ''section'' nroff cmd
shell	sh	Str	sh="/bin/sh"	full path name of the shell
showmatch	sm	Bool	nosm	show matching ()[]{}

showmode	smd	Bool	nosmd	say when in input mode
shiftwidth	sw	Num	sw=8	shift amount for < and >
sidescroll	ss	Num	ss=8	amount of sideways scrolling
sync	sy	Bool	nosy	call *sync*() often
tabstop	ts	Num	ts=8	width of tab characters
term	te	Str	te="$TERM"	name of the termcap entry
vbell	vb	Bool	vb	use visible alternative to bell
warn	wa	Bool	wa	warn for ! if file modified
wrapmargin	wm	Num	wm=0	wrap long lines in input mode
wrapscan	ws	Bool	ws	at EOF searches wrap to line 1

There are three types of options: Bool, string, and numeric. Boolean options are made TRUE by giving the name of the option as an argument to the ‘‘set’’ command; they are made FALSE by prefixing the name with ‘‘no’’. For example, ‘‘set autoindent’’ makes the autoindent option TRUE, and ‘‘set noautoindent’’ makes it FALSE.

To change the value of a string or numeric option, pass the ‘‘set’’ command the name of the option, followed by an ‘‘=’’ sign and the option’s new value. For example, ‘‘set tabstop=8’’ will give the tabstop option a value of 8. For string options, you may enclose the new value in quotes.

5.3.5.1. AutoIndent

During input mode, the autoindent option will cause each added line to begin with the same amount of leading white space as the line above it. Without autoindent, added lines are initially empty.

5.3.5.2. AutoPrint

This option only affects *ex* mode. If the autoprint option on, and either the cursor has moved to a different line or the previous command modified the file, then *elvis* will print the current line.

5.3.5.3. AutoWrite

When you are editing one file and decide to switch to another − via the :tag command, or :next command, perhaps − if your current file has been modified, then *elvis* will normally print an error message and refuse to switch.

However, if the autowrite option is on, then *elvis* will write the modified version of the current file and successfully switch to the new file.

5.3.5.4. CC

The :cc command runs the C compiler. This option should be set to the name of your compiler.

5.3.5.5. CharAttr

Many text formatting programs allow you to designate portions of your text to be underlined, italicized, or boldface by embedding the special strings \fU, \fI, and \fB in your text. The special string \fR marks the end of underlined or boldface text.

Elvis normally treats those special strings just like any other text.

However, if the charattr option is on, then *elvis* will interpret those special strings correctly, to display underlined or boldface text on the screen. (This only works, of course, if your terminal can display underlined and boldface, and if the termcap entry says how to do it.)

5.3.5.6. COlumns

This is a ‘‘read only’’ option. You cannot change its value, but you can have *elvis* print it. It shows how wide your screen is.

5.3.5.7. DIGraph

This option is used to enable/disable recognition of digraphs. The default value is nodigraph, which means that digraphs will not be recognized.

5.3.5.8. DIRectory

Elvis stores text in temporary files. This option allows you to control which directory those temporary files will appear in. The default is */usr/tmp*.

This option can only be set in a *.exrc* file; after that, *elvis* will have already started making temporary files in some other directory, so it would be too late.

5.3.5.9. EDcompatible

This option affects the behavior of the ‘‘:s/regexp/text/options’’ command. It is normally off (:se noed) which causes all of the substitution options to be off unless explicitly given.

However, with *edcompatible* on (:se ed), the substitution command remembers which options you used last time. Those same options will continue to be used until you change them. In edcompatible mode, when you explicitly give the name of a substitution option, you will toggle the state of that option.

5.3.5.10. ErrorBells

Elvis normally rings a bell when you do something wrong. This option lets you disable the bell.

5.3.5.11. ExRefresh

The *ex* mode of *elvis* writes many lines to the screen. You can make *elvis* either write each line to the screen separately, or save up many lines and write them all at once.

The exrefresh option is normally on, so each line is written to the screen separately.

You may wish to turn the exrefresh option off (:se noer) if the ‘‘write’’ system call is costly on your machine, or if you are using a windowing environment. (Windowing environments scroll text a lot faster when you write many lines at once.)

This option has no effect in visual command mode or input mode.

5.3.5.12. FlipCase

The flipcase option allows you to control how the non-ASCII characters are altered by the ‘‘~’’ command.

The string is divided into pairs of characters. When ‘‘~’’ is applied to a non-ASCII character, *elvis* looks up the character in the flipcase string to see which pair it is in, and replaces it by the other character of the pair.

5.3.5.13. HideFormat

Many text formatters require you to embed format commands in your text, on lines that start with a ‘‘.’’ character. *Elvis* normally displays these lines like any other text, but if the hideformat option is on, then format lines are displayed as blank lines.

5.3.5.14. IgnoreCase

Normally, when *elvis* searches for text, it treats uppercase letters as being different for lowercase letters.

When the ignorecase option is on, uppercase and lowercase are treated as equal.

5.3.5.15. InputMode

This option allows you to have *elvis* start up in insert mode. You can still exit insert mode at any time by hitting the ESC key, as usual. Usually, this option would be set in your *.exrc* file.

5.3.5.16. KeyTime

The arrow keys of most terminals send a multi-character sequence. It takes a measurable amount of time for these sequences to be transmitted. The keytime option allows you to control the maximum amount of time to allow for an arrow key (or other mapped key) to be received in full.

The default keytime value is 2. Because of the way UNIX time-keeping works, the actual amount of time allowed will vary slightly, but it will always be between 1 and 2 seconds.

If you set keytime to 1, then the actual amount of time allowed will be between 0 and 1 second. This will generally make the keyboard’s response be a little faster (mostly for the ESC key), but on those occasions where the time allowed happens to be closer to 0 than 1 second, *elvis* may fail to allow enough time for an arrow key’s sequence to be received fully.

As a special case, you can set keytime to 0 to disable this time limit stuff altogether. The big problem here is: If your arrow keys’ sequences start with an ESC, then every time you hit your ESC key *elvis* will wait... and wait... to see if maybe that ESC was part of an arrow key’s sequence.

NOTE: this option is a generalization of the timeout option of the real *vi*.

5.3.5.17. KeywordPrg

Elvis has a special keyword lookup feature. You move the cursor onto a word, and hit shift−K, and *elvis* uses another program to look up the word and display information about it.

This option says which program gets run.

The default value of this option is ‘‘ref’’, which is a program that looks up the definition of a function in C. It looks up the function name in a file called ‘‘refs’’ which is created by ctags.

You can substitute other programs, such as an English dictionary program or the on-line manual. *Elvis* runs the program, using the keyword as its only argument. The program should write information to *stdout*. The program’s exit status should be 0, unless you want *elvis* to print ‘‘<<< failed >>>’’.

5.3.5.18. LiNes

This ‘‘read only’’ option shows how many lines you screen has.

5.3.5.19. LIst

In nolist mode (the default), *elvis* displays text in a ‘‘normal’’ manner — with tabs expanded to an appropriate number of spaces, etc.

However, sometimes it is useful to have tab characters displayed differently. In list mode, tabs are displayed as ‘‘^I’’, and a ‘‘$’’ is displayed at the end of each line.

5.3.5.20. MAgic

The search mechanism in *elvis* can accept ‘‘regular expressions’’ — strings in which certain characters have special meaning.

The magic option is normally on, which causes these characters to be treated specially.

If you turn the magic option off (:se noma), then all characters except ^ and $ are treated literally. ^ and $ retain their special meanings regardless of the setting of magic.

5.3.5.21. MaKe

The :make command runs your ‘‘make’’ program. This option defines the name of your ‘‘make’’ program.

5.3.5.22. ModeLine

Elvis supports modelines. Modelines are lines near the beginning or end of your text file which contain ‘‘ex:yowza:’’, where ‘‘yowza’’ is any *ex* command. A typical ‘‘yowza’’ would be something like ‘‘set ts=4 ca kp=spell’’.

Normally these lines are ignored, for security reasons, but if you have ‘‘set modeline’’ in your *.exrc* file then ‘‘yowza’’ is executed.

5.3.5.23. PAragraphs

The { and } commands move the cursor forward or backward in increments of one paragraph. Paragraphs may be separated by blank lines, or by a ‘‘dot’’ command of a text formatter. Different text formatters use different ‘‘dot’’ commands. This option allows you to configure *elvis* to work with your text formatter.

It is assumed that your formatter uses commands that start with a ‘‘.’’ character at the front of a line, and then have a one- or two-character command name.

The value of the paragraphs option is a string in which each pair of characters is one possible form of your text formatter’s paragraph command.

5.3.5.24. ReadOnly

Normally, *elvis* will let you write back any file to which you have write permission. If you do not have write permission, then you can only write the changed version of the file to a **different** file.

If you set the readonly option, then *elvis* will pretend you do not have write permission to **any** file you edit. It is useful when you really only mean to use *elvis* to look at a file, not to change it. This way you cannot change it accidentally.

This option is normally off, unless you use the ‘‘view’’ alias of *elvis*. *View* is like *vi* except that the readonly option is on.

5.3.5.25. REport

Commands in *elvis* may affect many lines. For commands that affect a lot of lines, *elvis* will output a message saying what was done and how many lines were affected. This option allows you to define what ‘‘a lot of lines’’ means. The default is 5, so any command which affects 5 or more lines will cause a message to be shown.

5.3.5.26. SCroll

The ^U and ^D keys normally scroll backward or forward by half a screen full, but this is adjustable. The value of this option says how many lines those keys should scroll by.

5.3.5.27. SEctions

The [[and]] commands move the cursor backward or forward in increments of 1 section. Sections may be delimited by a { character in column 1 (which is useful for C source code) or by means of a text formatter’s ‘‘dot’’ commands.

This option allows you to configure *elvis* to work with your text formatter’s ‘‘section’’ command, in exactly the same way that the paragraphs option makes it work with the formatter’s ‘‘paragraphs’’ command.

5.3.5.28. SHell

When *elvis* forks a shell (perhaps for the :! or :shell commands) this is the program that is uses as a shell. This is ‘‘/bin/sh’’ by default, unless you have set the `SHELL` (or `COMSPEC`, for MS-DOS) environment variable, it which case the default value is copied from the environment.

5.3.5.29. ShiftWidth

The < and > commands shift text left or right by some uniform number of columns. The shiftwidth option defines that ‘‘uniform number’’. The default is 8.

5.3.5.30. ShowMatch

With showmatch set, in input mode every time you hit one of)}], *elvis* will momentarily move the cursor to the matching ({[and then return to the original position. If there is no matching ({[then it will beep at you.

5.3.5.31. ShowMoDe

In visual mode, it is easy to forget whether you are in the visual command mode or input/replace mode. Normally, the showmode option is off, and you have not a clue as to which mode you are in. If you turn the showmode option on, though, a little message will appear in the lower right-hand corner of your screen, telling you which mode you are in.

5.3.5.32. SideScroll

For long lines, *elvis* scrolls sideways. (This is different from the real *vi*, which wraps a single long line onto several rows of the screen.)

To minimize the number of scrolls needed, *elvis* moves the screen sideways by several characters at a time. The value of this option says how many characters' widths to scroll at a time.

Generally, the faster your screen can be redrawn, the lower the value you will want in this option.

5.3.5.33. SYnc

If the system crashes during an edit session, then most of your work can be recovered from the temporary file that *elvis* uses to store changes. However, sometimes the OS will not copy changes to the hard disk immediately, so recovery might not be possible. The [no]sync option lets you control this.

In nosync mode (which is the default, for UNIX), *elvis* lets the operating system control when data is written to the disk. This is generally faster.

In sync mode (which is the default, for MS-DOS), *elvis* forces all changes out to disk every time you make a change. This is generally safer, but slower. It can also be a rather rude thing to do on a multi-user system.

5.3.5.34. TabStop

Tab characters are normally 8 characters wide, but you can change their widths by means of this option.

5.3.5.35. TErm

This ''read only'' option shows the name of the termcap entry that *elvis* is using for your terminal.

5.3.5.36. VBell

If your termcap entry describes a visible alternative to ringing your terminal's bell, then this option will say whether the visible version gets used or not. Normally it will be.

If your termcap does **not** include a visible bell capability, then the vbell option will be off, and you cannot turn it on.

5.3.5.37. WArn

If you have modified a file but not yet written it back to disk, then *elvis* will normally print a warning before executing a ‘‘:!cmd’’ command. However, in nowarn mode, this warning is not given.

Elvis also normally prints a message after a successful search that wrapped at EOF. The [no]warn option can also disable this warning.

5.3.5.38. WrapMargin

Normally (with wrapmargin=0) *elvis* will let you type in extremely long lines, if you wish.

However, with wrapmargin set to something other that 0 (wrapmargin=10 is nice), *elvis* will automatically cause long lines to be ‘‘wrapped’’ on a word break for lines longer than wrapmargin’s setting.

5.3.5.39. WrapScan

Normally, when you search for something, *elvis* will find it no matter where it is in the file. *Elvis* starts at the cursor position, and searches forward. If *elvis* hits EOF without finding what you are looking for, then it wraps around to continue searching from line 1.

If you turn off the wrapscan option (:se nows), then when *elvis* hits EOF during a search, it will stop and say so.

5.3.6. CUT BUFFERS

When *elvis* deletes text, it stores that text in a cut buffer. This happens in both visual mode and *ex* mode. There is no practical limit to how much text a cut buffer can hold.

There are 36 cut buffers: 26 named buffers ("a through "z), 9 anonymous buffers ("1 through "9), and 1 extra cut buffer (".).

In *ex* mode, the :move and :copy commands use a cut buffer to temporarily hold the text to be moved/copied.

5.3.6.1. Filling

In visual mode, text is copied into a cut buffer when you use the d, y, c, C, or s commands.

By default, the text goes into the "1 buffer. The text that used to be in "1 gets shifted into "2, "2 gets shifted into "3, and so on. The text that used to be in "9 is lost. This way, the last 9 things you deleted are still accessible.

You can also put the text into a named buffer — "a through "z. To do this, you should type the buffer’s name (two keystrokes: a double-quote and a lowercase letter) before the d/y/c/C/s command. When you do this, "1 through "9 are not affected by the cut.

You can append text to one of the named buffers. To do this, type the buffer’s name in uppercase (a double-quote and an uppercase letter) before the d/y/c/C/s command.

The ". buffer is special. It is not affected by the d/y/c/C/s command. Instead, it stores the text that you typed in the last time you were in input mode. It is used to implement the ‘‘.’’ visual command, and ^A in input mode.

In *ex* mode (also known as colon mode), the :delete, :change, and :yank commands all copy text into a cut buffer. Like the visual commands, these *ex* commands normally use the "1

buffer, but you can use one of the named buffers by giving its name after the command. For example,

```
:20,30y a
```

will copy lines 20 through 30 into cut buffer "a.

You cannot directly put text into the ". buffer, or the "2 through "9 buffers.

5.3.6.2. Pasting from a Cut Buffer

There are two styles of pasting: line-mode and character-mode. If a cut buffer contains whole lines (from a command like ‘‘dd’’) then line-mode pasting is used; if it contains partial lines (from a command like ‘‘dw’’) then character-mode pasting is used. The *ex* commands always cut whole lines.

Character-mode pasting causes the text to be inserted into the line that the cursor is on.

Line-mode pasting inserts the text on a new line above or below the line that the cursor is on. It does not affect the cursor’s line at all.

In visual mode, the p and P commands insert text from a cut buffer. Uppercase P will insert it before the cursor, and lowercase p will insert it after the cursor. Normally, these commands will paste from the "1 buffer, but you can specify any other buffer to paste from. Just type its name (a double-quote and another character) before you type the P or p.

In *ex* mode, the (pu)t command pastes text after a given line. To paste from a buffer other that "1, enter its name after the command.

5.3.6.3. Macros

The contents of a named cut buffer can be executed as a series of *ex/vi* commands.

To put the instructions into the cut buffer, you must first insert them into the file, and then delete them into a named cut buffer.

To execute a cut buffer’s contents as *ex* commands, you should give the *ex* command ‘‘@’’ and the name of the buffer. For example, :@z will execute "z as a series of *ex* commands.

To execute a cut buffer’s contents as visual commands, you should give the visual command ‘‘@’’ and the letter of the buffer’s name. The visual ‘‘@’’ command is different from the *ex* ‘‘@’’ command. They interpret the cut buffer’s contents differently.

The visual @ command can be rather finicky. Each character in the buffer is interpreted as a keystroke. If you load the instructions into the cut buffer via a "zdd command, then the newline character at the end of the line will be executed just like any other character, so the cursor would be moved down 1 line. If you do not want the cursor to move down 1 line at the end of each @z command, then you should load the cut buffer by saying 0"zD instead.

Although cut buffers may hold any amount of text, *elvis* can only *execute* small buffers. For *ex* mode, the buffer is limited to about 1k bytes. For visual mode, the buffer is limited to about 80 bytes. If a buffer is too large to execute, an error message is displayed.

You cannot nest @ commands. You cannot run @ commands from your *.exrc* file, or any other :source file either. Similarly, you cannot run a :source command from within an @ command. Hopefully, these restrictions will be lifted in a later version.

5.3.6.4. The Effect of Switching Files

When *elvis* first starts up, all cut buffers are empty. When you switch to a different file (via the :n or :e commands perhaps) the 9 anonymous cut buffers are emptied again, but the other 27 buffers retain their text.

5.3.7. DIFFERENCES BETWEEN ELVIS & BSD VI/EX

5.3.7.1. Extensions

```
:mkexrc
:mk
```

This *ex* command saves the current :set and :map configurations in the *.exrc* file in your current directory.

```
:Next
:previous
:N
:pre
```

These commands move backwards through the args list.

```
zz
```

In visual command mode, the (lowercase) ‘‘zz’’ command will center the current line on the screen, like ‘‘z=’’.

The default count value for ‘‘.’’ is the same as the previous command which ‘‘.’’ is meant to repeat. However, you can supply a new count if you wish. For example, after ‘‘3dw’’, ‘‘.’’ will delete 3 words, but ‘‘5.’’ will delete 5 words.

```
".
```

The text which was most recently input (via a ‘‘cw’’ command, or something similar) is saved in a cut buffer called ". (which is a pretty hard name to write in an English sentence).

```
K
```

In visual command mode, you can move the cursor onto a word and press shift−K to have *elvis* run a reference program to look that word up. This command alone is worth the price of admission! See the ctags and ref programs.

```
#
```

In visual command mode, you can move the cursor onto a number and then hit ## or #+ to increment that number by 1. To increment it by a larger amount, type in the increment value before hitting the initial #. The number can also be decremented or set by hitting #− or #=, respectively.

```
input
```

You can backspace past the beginning of the line. The arrow keys work in input mode.

If you type CTRL−A, then the text that you input last time is inserted. You will remain in

input mode, so you can backspace over part of it, or add more to it. (This is sort of like CTRL−@ on the real *vi*, except that CTRL−A really works.)

CTRL−P will insert the contents of the cut buffer.

Real *vi* can only remember up to 128 characters of input, but *elvis* can remember any amount.

The ^T and ^D keys can adjust the indent of a line no matter where the cursor happens to be in that line.

You can save your file and exit *elvis* directly from input mode by hitting CTRL−Z twice.

Elvis supports digraphs as a way to enter non-ASCII characters.

```
:set inputmode
:se im
```

If you set this flag in your *.exrc* file, then *elvis* will start up in input mode instead of visual command mode.

```
:set charattr
:se ca
```

Elvis can display ‘‘backslash−f’’ style character attributes on the screen as you edit. The following example shows the recognized attributes:

normal \fBboldface\fR \fIitalics\fR \fUunderlined\fR normal

NOTE: you must compile *elvis* without the `-DNO_CHARATTR` flag for this to work.

```
:set sync
:se sy
```

After a crash, you can usually recover the altered form of the file from the temporary file that *elvis* uses. With the sync option turned on, the odds are shifted a little more in your favor because *elvis* will perform a *sync*() call after each change has been written to the temporary file.

```
cursor shape
```

If your terminal’s termcap entry includes the necessary capabilities, *elvis* changes the shape of the cursor to indicate which mode you are in.

```
:set hideformat
:se hf
```

This option hides format control lines. (They are displayed on the screen as blank lines.)

```
:errlist
*
elvis -m
```

Elvis is clever enough to parse the error messages emitted by many compilers. To use this feature, you should collect your compiler’s error messages into a file called *errlist*; *elvis* will read this file, determine which source file caused the error messages, start editing that file, move the cursor to the line where the error was detected, and display the error message on the status line.

5.3.7.2. Omissions

The replace mode is a hack. It does not save the text that it overwrites.

Long lines are displayed differently — where the real *vi* would wrap a long line onto several rows of the screen, *elvis* simply displays part of the line, and allows you to scroll the screen sideways to see the rest of it.

The ‘‘:preserve’’ and ‘‘:recover’’ commands are missing. So is the **−r** flag. Since ‘‘:recover’’ is used so rarely it was implemented as a separate program. There’s no need to load the recovery code into memory every time you edit a file.

LISP support is missing.

Due to naming conventions used for the temporary files, *elvis* can be creating no more that one new file per directory at any given time. Any number of existing files can be edited at the same time on multitasking computer systems, but only one new file can be created simultaneously per directory. To relieve this problem, you would have to edit tmp.c and virec.c This is expected to be done in version 1.5

Autoindent mode acts a little different from the real *vi*. It is still quite useful, but if you frequently use both *vi* and *elvis* then the differences may be annoying. Autoindent is **gradually** improving.

The visual ‘‘put’’ command cannot be repeated by hitting the . key.

5.3.8. INTERNAL

You do not need to know the material in this section to use *elvis*. You only need it if you intend to modify *elvis*.

5.3.8.1. The temporary file

The temporary file is divided into blocks of 1024 bytes each.

When *elvis* starts up, the file is copied into the temporary file. Small amounts of extra space are inserted into the temporary file to insure that no text lines cross block boundaries; this speeds up processing and simplifies storage management. The ‘‘extra space‘‘ is filled with NUL characters; the input file must not contain any NULs, to avoid confusion.

The first block of the temporary file is an array of shorts which describe the order of the blocks; that is, header[1] is the block number of the first block, and so on. This limits the temporary file to 512 active blocks, so the largest file you can edit is about 400K bytes long!

When blocks are altered, they are rewritten to a **different** block in the file, and the in-core version of the header block is updated accordingly. The in-core header block will be copied to the temp file immediately before the next change... or, to undo this change, swap the old header (from the temp file) with the new (in-core) header.

Elvis maintains another in-core array which contains the line-number of the last line in every block. This allows you to go directly to a line, given its line number.

5.3.8.2. Implementation of Editing

There are three basic operations which affect text:

- delete text − delete(from, to)
- add text − add(at, text)
- yank text − cut(from, to)

To yank text, all text between two text positions is copied into a cut buffer. The original text is not changed. To copy the text into a cut buffer, you need only remember which physical blocks that contain the cut text, the offset into the first block of the start of the cut, the offset into the last block of the end of the cut, and what kind of cut it was. (Cuts may be either character cuts or line cuts; the kind of a cut affects the way it is later ‘‘put’’.) This is implemented in the function *cut*()

To delete text, you must modify the first and last blocks, and remove any reference to the intervening blocks in the header’s list. The text to be deleted is specified by two marks. This is implemented in the function *delete*().

To add text, you must specify the text to insert (as a null-terminated string) and the place to insert it (as a mark). The block into which the text is to be inserted may need to be split into as many as four blocks, with new intervening blocks needed as well... or it could be as simple as modifying a single block. This is implemented in the function *add*().

Other interesting functions are *paste*() (to copy text from a cut buffer into the file), *modify*() (for an efficient way to implement a combined delete/add sequence), and *input*() (to get text from the user and insert it into the file).

When text is modified, an internal file-revision counter, called ‘‘changes’’, is incremented. This counter is used to detect when certain caches are out of date. (The ‘‘changes’’ counter is also incremented when we switch to a different file, and also in one or two similar situations — all related to invalidating caches.)

5.3.8.3. Marks and the Cursor

Marks are places within the text. They are represented internally as a long variable which is split into two bit-fields: a line number and a character index. Line numbers start with 1, and character indexes start with 0.

Since line numbers start with 1, it is impossible for a set mark to have a value of 0L. 0L is therefore used to represent unset marks.

When you do the ‘‘delete text’’ change, any marks that were part of the deleted text are unset, and any marks that were set to points after it are adjusted. Similarly, marks are adjusted after new text is inserted.

The cursor is represented as a mark.

5.3.8.4. Colon Command Interpretation

Colon commands are parsed, and the command name is looked up in an array of structures which also contain a pointer to the function that implements the command, and a description of the arguments that the command can take. If the command is recognized and its arguments are legal, then the function is called.

Each function performs its task; this may cause the cursor to be moved to a different line, or

whatever.

5.3.8.5. Screen Control

The screen is updated via a package which looks like the ‘‘curses’’ library, but is not. It is actually much simpler. Most curses operations are implemented as macros which copy characters into a large I/O buffer, which is then written with a single large *write*() call as part of the *refresh*() operation.

The functions which modify text (namely *add*() and *delete*()) remember where text has been modified. They do this by calling the function *redrawrange*(). The screen redrawing function, *redraw*(), uses these clues to help it reduce the amount of text that is redrawn each time.

5.3.8.6. Portability

To improve portability, *elvis* collects as many of the system-dependent definitions as possible into the config.h file. This file begins with some preprocessor instructions which attempt to determine which compiler and operating system you have. After that, it conditionally defines some macros and constants for your system.

One of the more significant macros is *ttyread(buf,*n) This macro is used to read raw characters from the keyboard. An attempt to read may be cut short by a `SIGALRM` signal. For UNIX systems, this simply reads bytes from *stdin*. For MSDOS, TOS, and OS9, *ttyread*() is a function defined in *curses.c*. There is also a *ttywrite*() macro.

The *tread*() and *twrite*() macros are versions of *read*() and *write*() that are used for text files. On UNIX systems, these are equivalent to *read*() and *write*(). On MS-DOS, these are also equivalent to *read*() and *write*(), since DOS libraries are generally clever enough to convert newline characters automatically. For Atari TOS, though, the MWC library is too stupid to do this, so we had to do the conversion explicitly.

Other macros may substitute *index*() for *strchr*(), or *bcopy*() for *memcpy*(), or map the ‘‘void’’ data type to ‘‘int’’, or whatever.

The file *tinytcap.c* contains a set of functions that emulate the termcap library for a small set of terminal types. The terminal-specific info is hard-coded into this file. It is only used for systems that do not support real termcap. Another alternative for screen control can be seen in the *curses.h* and *pc.c* files. Here, macros named `VOIDBIOS` and `CHECKBIOS` are used to indirectly call functions which perform low-level screen manipulation via BIOS calls.

The *stat*() function must be able to come up with UNIX-style major/minor/inode numbers that uniquely identify a file or directory.

Please try to keep you changes localized, and wrap them in #if/#endif pairs, so that *elvis* can still be compiled on other systems. And please forward updates to the author so that they can be incorporated into the latest-and-greatest version of *elvis*.

5.3.9. CFLAGS

Elvis uses many preprocessor symbols to control compilation. Some of these control the sizes of buffers and such. The `-DNO_XXXX` options remove small sets of related features.

Most *elvis* users will probably want to keep all features available. Minix-PC users, though, will have to sacrifice some sets because otherwise *elvis* would be too bulky to compile. The

‘‘asld’’ phase of the compiler craps out.

−DM_SYSV, −DTOS, −DCOHERENT
: These flags tell the compiler that *elvis* is being compiled for System-V UNIX, Atari TOS, or Coherent, respectively. For other systems, the config.h file can generally figure it out automatically.

−DDATE=*string*
: DATE should be defined to be a string constant. It is printed by the :version command as the compilation date of the program.

  It is only used in *cmd1.c*, and even there you may leave it undefined without causing an urp.

−DNBUFS=*numberElvis* keeps most of your text in a temporary file; only a small amount is actually stored in RAM. This flag allows you to control how much of the file can be in RAM at any time. The default is 5 blocks. (See the −DBLKSIZE flag, below.)

  More RAM allows global changes to happen a little faster. If you are just making many small changes in one section of a file, though, extra RAM will not help much.

−DBLKSIZE=*number*
: This controls the size of blocks that *elvis* uses internally. The value of BLKSIZE must be a power of two. The default value is 1024, which allows you to edit files up to almost 512K bytes long. Every time you double BLKSIZE, you quadruple the size of a text file that *elvis* can handle, but you also cause the temporary file to grow faster.

−DTMPDIR=*string*
: This sets the default value of the ‘‘directory’’ option, which specifies where the temporary files should reside. The value of TMPDIR must be a string, so be sure your value includes the quote characters on each end.

−DEXRC=*str*, −DHMEXRC=*str*, −DSYSEXRC=*str*, −DEXINIT=*str*
: This lets you control the names of the initialization files. Their values must be strings, so be careful about quoting.

  EXRC is the name of the initialization file in the current directory. Its default value is *.exrc* on Amoeba. For other systems, check the config.h file.

  HMEXRC is the name of the initialization file in your home directory. By default, it is the same as EXRC. *Elvis* will automatically prepend the name of your home directory to HMEXRC at run time, so do not give a full path name.

  SYSEXRC is the name of a system-wide initialization file. It has no default value; if you do not define a value for it, then the code that supports SYSEXRC just is not compiled. The value of SYSEXRC should be a full path name, in quotes.

  EXINIT is the name of an environment variable that can contain initialization commands. Normally, its value is ‘‘EXINIT’’.

−DKEYWORDPRG=*string*
: This flag determines the default value of the ‘‘keywordprg’’ option. Its value must be a string, so be careful about quoting. The default value of this flag is ‘‘ref’’, which is a C reference program.

−DCC_COMMAND=*string* −DMAKE_COMMAND=*string* −DERRLIST=*string*

These control the names of the C compiler, the *make* utility, and the error output file, respectively. They are only used if −DNO_ERRLIST is not given.

−DMAXMAPS=*number*
This controls the capacity of the key map table.

−DMAXRCLEN=*number*
This determines how large a *.exrc* file can be (measured in bytes). The default is 1000 bytes. If you increase this value significantly, then you may need to allocate extra memory for the stack. See the CHMEM setting in the *Makefile*.

−DSHELL=*string*
This is the default value of the ‘‘shell’’ option, and hence the default shell used from within *elvis*. This only controls the default; the value you give here may be overridden at run-time by setting an environment variable named SHELL (or COMSPEC for MS-DOS). Its value must be a string constant, so be careful about quoting.

−DTAGS=*string*
This sets the name of the ‘‘tags’’ file, which is used by the :tag command. Its value must be a string constant, so be careful about quoting.

−DCS_IBMPC
The digraph table and flipcase option will normally start out empty. However, if you add −DCS_IBMPC or −DCS_LATIN1 to your CFLAGS, then they will start out filled with values that are appropriate for the IBM PC character set or the ISO Latin−1 character set, respectively.

−DDEBUG
This adds the ‘‘:debug’’ and ‘‘:validate’’ commands, and also adds many internal consistency checks. It increases the size of the ‘‘.text’’ segment by about 6K.

−DCRUNCH
This flag removes some non-critical code, so that *elvis* is smaller. For example, it removes a short-cut from the regexp package, so that text searches are slower. Also, screen updates are not as efficient. A couple of obscure features are disabled by this, too.

−DNO_MKEXRC
This removes the ‘‘:mkexrc’’ command, so you have to create any *.exrc* files manually. The size of the .text segment will be reduced by about 600 bytes.

−DNO_CHARATTR
Permanently disables the charattr option. This reduces the size of your ‘‘.text’’ segment by about 850 bytes.

−DNO_RECYCLE
Normally, *elvis* will recycle space (from the tmp file) which contains totally obsolete text. This flag disables this recycling. Without recycling, the ‘‘.text’’ segment is about 1K smaller than it would otherwise be, but the tmp file grows much faster. If you have a lot of free space on your hard disk, but *elvis* is too bulky to run with recycling, then try it without recycling.

When using a version of *elvis* that has been compiled with −DNO_RECYCLE, you should be careful to avoid making many small changes to a file because each individual change will cause the tmp file to grow by at least 1k. Hitting ‘‘x’’ thirty times counts as

thirty changes, but typing ‘‘30x’’ counts as one change. Also, you should occasionally do a ‘‘:w’’ followed by a ‘‘:e’’ to start with a fresh tmp file.

−DNO_SENTENCE
Leaves out the ‘‘(’’ and ‘‘)’’ visual mode commands. Also, the ‘‘[[’’, ‘‘]]’’, ‘‘{’’, and ‘‘}’’ commands will not recognize *roff macros. The sections and paragraphs options go away. This saves about 650 bytes in the ‘‘.text’’ segment.

−DNO_CHARSEARCH
Leaves out the visual commands which locate a given character in the current line: ‘‘f’’, ‘‘t’’, ‘‘F’’, ‘‘T’’, ‘‘,’’ and ‘‘;’’. This saves about 900 bytes.

−DNO_EXTENSIONSLeaves out the ‘‘K’’ and ‘‘#’’ visual commands. Also, the arrow keys will no longer work in input mode. (Other extensions are either inherent in the design of *elvis*, or are controlled by more specific flags, or are too tiny to be worth removing.) This saves about 250 bytes.

−DNO_MAGIC
Permanently disables the ‘‘magic’’ option, so that most meta-characters in a regular expression are **not** recognized. This saves about 3k of space in the ‘‘.text’’ segment, because the complex regular expression code can be replaced by much simpler code.

−DNO_SHOWMODE
Permanently disables the ‘‘showmode’’ option, saving about 250 bytes.

Normally, *elvis* tries to adjust the shape of the cursor as a reminder of which mode you are in. The −DNO_CURSORSHAPE flag disables this, saving about 150 bytes.

−DNO_DIGRAPH
To allow entry of non-ASCII characters, *elvis* supports digraphs. A digraph is a single (non-ASCII) character which is entered as a combination of two other (ASCII) characters. If you do not need to input non-ASCII characters, or if your keyboard supports a better way of entering non-ASCII characters, then you can disable the digraph code and save about 450 bytes.

−DNO_ERRLIST
Elvis adds a ‘‘:errlist’’ command, which is useful to programmers. If you do not need this feature, you can disable it via the −DNO_ERRLIST flag. This will reduce the .text segment by about 900 bytes, and the .bss segment by about 300 bytes.

−DNO_ABBR
The −DNO_ABBR flag disables the ‘‘:abbr’’ command, and reduces the size of *elvis* by about 600 bytes.

−DNO_OPTCOLSWhen *elvis* displays the current options settings via the ‘‘:set’’ command, the options are normally sorted into columns. The −DNO_OPTCOLS flag causes the options to be sorted across the rows, which is much simpler. The −DNO_OPTCOLS flag will reduce the size of your .text segment by about 500 bytes.

−DNO_MODELINE
This removes all support for modelines.

5.3.10. TERMCAP

Elvis uses fairly standard termcap fields for most things. The cursor shape names are non-standard but other than that there should be no surprises.

5.3.10.1. Required numeric fields

:co#: number of columns on the screen (characters per line)
:li#: number of lines on the screen

5.3.10.2. Required string fields

:ce=: clear to end-of-line
:cl=: home the cursor & clear the screen
:cm=: move the cursor to a given row/column
:up=: move the cursor up one line

5.3.10.3. Boolean fields

:am: auto margins − wrap when a char is written to the last column?
:pt: physical tabs?

5.3.10.4. Optional string fields

:al=: insert a blank row on the screen
:dl=: delete a row from the screen
:cd=: clear to end of display
:ei=: end insert mode
:ic=: insert a blank character
:im=: start insert mode
:dc=: delete a character
:sr=: scroll reverse (insert a row at the top of the screen)
:vb=: visible bell
:ti=: terminal initialization string, to start full-screen mode
:te=: terminal termination, to end full-screen mode
:ks=: enables the cursor keypad
:ke=: disables the cursor keypad

5.3.10.5. Optional strings received from the keyboard

:kd=: sequence sent by the <down arrow> key
:kl=: sequence sent by the <left arrow> key
:kr=: sequence sent by the <right arrow> key
:ku=: sequence sent by the <up arrow> key
:kP=: sequence sent by the <PgUp> key
:kN=: sequence sent by the <PgDn> key
:kh=: sequence sent by the <Home> key
:kH=: sequence sent by the <End> key

Originally, termcap did not have any names for the <PgUp>, <PgDn>, <Home>, and <End> keys. Although the capability names shown in the table above are the most common, they are **not** universal. SCO Xenix uses :PU=:PD=:HM=:EN=: for those keys. Also, if the four

arrow keys happen to be part of a 3x3 keypad, then the five non-arrow keys may be named :K1=: through :K5=:, so an IBM PC keyboard may be described using those names instead. *Elvis* can recognize any of these names.

5.3.10.6. Optional fields that describe character attributes

:so=: :se=: start/end standout mode (We do not care about :sg#:)
:us=: :ue=: start/end underlined mode
:md=: :me=: start/end boldface mode
:as=: :ae=: start/end alternate character set (italics)
:ug#: visible gap left by :us=:ue=:md=:me=:as=:ae=:

5.3.10.7. Optional fields that affect the cursor's shape

The :cQ=: string is used by *elvis* immediately before exiting to undo the effects of the other cursor shape strings. If :cQ=: is not given, then all other cursor shape strings are ignored.

:cQ=: normal cursor
:cX=: cursor shape used for reading *ex* command — steady underline
:cV=: cursor shape used for reading *vi* commands — steady block
:cI=: cursor shape used during *vi* input mode — blinking underline
:cR=: cursor shape used during *vi* replace mode — blinking block

If the capabilities above are not given, then *elvis* will try to use the following values instead.

:ve=: normal cursor, used as :cQ=:cX=:cI=:cR=:
:vs=: gaudy cursor, used as :cV=:

5.3.11. ENVIRONMENT VARIABLES

Elvis examines several environment variables when it starts up. The values of these variables are used internally for a variety of purposes. You do not need to define all of these; on most systems, *elvis* only requires `TERM` to be defined. On MS-DOS systems, even that is optional.

5.3.11.1. TERM, TERMCAP

`TERM` tells *elvis* the name of the termcap entry to use. `TERMCAP` may contain either the entire termcap entry, or the full path name of the *termcap* file to search through.

5.3.11.2. TMP, TEMP

These only work for MS-DOS and Atari TOS. Either of these variables may be used to set the ‘‘directory’’ option, which controls where temporary files are stored. If you define them both, then `TMP` is used, and `TEMP` is ignored.

5.3.11.3. EXINIT

This variable may contain a colon-mode command, which will be executed after all of the *.exrc* files but before interactive editing begins.

5.3.11.4. SHELL, COMSPEC

You can use `COMSPEC` in MS-DOS, or `SHELL` in any other system, to specify which shell should be used for executing commands and expanding wildcards.

5.3.11.5. HOME

This variable should give the full path name of your home directory. *Elvis* needs to know the name of your home directory so it can locate the *.exrc* file there.

5.3.12. VERSIONS

Elvis currently works under BSD UNIX, AT&T System-V UNIX, SCO XENIX, Minix, Coherent, MS-DOS, Atari TOS, and OS9/68k. This section of the manual provides special information that applies to each particular version of *elvis*.

For all versions except MS-DOS, the file ‘‘Makefile.mix’’ should be copied to ‘‘Makefile’’, and then edited to select the correct set of options for your system. There is more information about this embedded in the file itself.

5.3.12.1. BSD UNIX

Temporary files are stored in */tmp*.

Elvis does not have an *expreserve* program yet. Instead, you should modify /etc/rc so that the temp files are not deleted when the system is rebooted. Find a line in /etc/rc which reads

```
rm -rf /tmp/*
```

or something like that, and change it to read

```
rm -rf /tmp/[^e]* /tmp/e[^l]* /tmp/el[^v]* /tmp/elv_*
```

If you do not have permission to modify */etc/rc*, then it would be unsafe for you to have *elvis* store any temp files in */tmp*. In this case, it is suggested that you make a ‘‘tmp’’ directory in your home directory and add ‘‘set directory=/your/home/directory/tmp’’ to your *.exrc* file or `EXINIT` variable.

Both *elvis* and the real *vi* read initialization commands from a file called *.exrc*, but the commands in that file might work on one but not the other. For example,

```
set keywordprg=/usr/myself/bin/ref
```

will work for *elvis*, but *vi* will complain because it does not have a *keywordprg* option. If the warning messages annoy you, then you can edit the config.h file to change the name of the initialization file *.exrc* to something else, such as *.elvisrc*.

If you use X windows, you may wish to add `-DCS_LATIN1` to CFLAGS. This will cause the digraph table and the flipcase option to have default values that are appropriate for the LATIN−1 character set. That is the standard character set for X.

5.3.12.2. System-V UNIX

If your system uses terminfo instead of termcap, then you will have to edit the `LIBS` setting in the *Makefile*. Currently it says `LIBS=-ltermcap`, but you may have to change it to `LIBS=-lterminfo` or `LIBS=-lterm` or something like that.

The */etc/rc* file should be modified, as described for BSD systems above. The potential

trouble with *.exrc* described above for BSD UNIX applies to System-V UNIX as well.

Elvis uses CTRL−C as the interrupt key, not Delete.

5.3.12.3. SCO Xenix

For Xenix-386, you can use the generic System-V settings. You may wish to add `−DCS_IBMPC` to CFLAGS, to have the digraph table and flipcase option start up in a mode that is appropriate for the console. There is a separate group of settings for use with Xenix-286. It already has `−DCS_IBMPC` in CFLAGS.

Because Xenix is so similar to System-V, everything said earlier about System-V applies to the Xenix version too.

5.3.12.4. MINIX

There are separate settings in Makefile.mix for MINIX-PC and MINIX-ST. The differences between these two are that the ST version uses ‘‘.o’’ for the object file extension where the PC version uses ‘‘.s’’, and the PC version has some extra flags in CFLAGS to reduce the size of *elvis*. The PC version also uses tinytcap (instead of the full termcap) to make it smaller.

MINIX-PC users should read the CFLAGS section of this manual very carefully.

The temporary files are stored in */usr/tmp*. The */usr/tmp* directory must exist before you run *elvis*, and it must be readable/writable by everybody. We use */usr/tmp* instead of */tmp* because after a system crash or power failure, you can recover the altered version of a file from the temporary file in */usr/tmp*. If it was stored in */tmp*, though, then it would be lost because */tmp* is probably located on the RAM disk.

Elvis uses CTRL−C as the interrupt key, not Delete.

5.3.12.5. Coherent

Elvis was ported to Coherent by Esa Ahola.

Elvis is too large to run under Coherent unless you eliminate some features via the CFLAGS setting. The recommended settings, in Makefile.mix, produce a working version of *elvis* which emulates *vi* faithfully, but lacks most of the extensions. You should read the CFLAGS section of this manual carefully.

You can probably reduce the size of *elvis* by using tinytcap.c instead of −lterm. This would allow you to keep most features of *elvis*, at the expense of terminal independence. (Tinytcap.c has ANSI escape sequences hard-coded into it.)

The temporary files are stored in /tmp. You should modify your /etc/rc file as described for BSD earlier.

5.3.12.6. MS-DOS

Elvis was ported to MS-DOS by Guntram Blohm and Martin Patzel.

There are two different Makefiles for *elvis* under MS-DOS. ‘‘Elvis.prj’’ should be used with Turbo-C, and ‘‘Elvis.mak’’ should be used with MSC and Microsoft MAKE. *Elvis* has been tested more thoroughly with MSC, so if you have a choice, use MSC.

Elvis stores its temporary files in C:\tmp. If this is not satisfactory, then you should edit the CFLAGS line of your *Makefile* to change `TMPDIR` to something else before compiling. You

can also control the name of the temp directory via an environment variable named TMP or TEMP. The directory must exist before you can run *elvis*.

Normally, the TERM environment variable should not be set, or else it should be set to ‘‘pcbios’’. This way, *elvis* will make calls to BIOS to update the screen. (If you do not like the colors that the BIOS interface uses, then edit the attr[] table in *pc.c.*)

You may prefer to use a device driver such as ANSI.SYS or NNANSI.SYS, for speed; or you may need to use a device driver for compatibility. If so, you should install one of these drivers by adding ‘‘driver = ansi.sys’’ to your CONFIG.SYS file, and then define TERM to be either ‘‘ansi’’ or ‘‘nansi’’ by adding a line such as ‘‘set TERM=ansi’’ to your AUTOEXEC.BAT file. You must then reboot for these changes to take effect. After that, *elvis* will notice the TERM setting and use the driver.

Under MS-DOS, *elvis* has an extra ‘‘:set’’ option called ‘‘pcbios’’ which indicates whether the BIOS is being used directly. This is a ‘‘read only’’ option; you cannot use it to switch your interface style in the middle of an edit session.

An extra program, called ‘‘wildcard’’, is needed for MS-DOS. It expands wildcard characters in file names.

5.3.12.7. Atari TOS

Elvis was ported to Atari TOS by Guntram Blohm and Martin Patzel. It is very similar to the MS-DOS version. It has only been tested with the Mark Williams C compiler.

The TERM environment variable is ignored; the ST port always assumes that TERM=vt52. The SHELL (not COMSPEC!) variable should be set to the name of a line-oriented shell.

A simple shell in included with *elvis*. Its source is in ‘‘shell.c’’, and the name of the executable is ‘‘shell.ttp’’. This was necessary because the standard Atari software does not offer any way to set environment variables. The file ‘‘profile.sh’’ should contain a set of instructions to be executed when the shell first starts up. An example of this file is included, but you will almost certainly want to edit it right away to match your configuration.

If you already have a command-line shell, then you’ll probably want to continue using it. The shell that comes with *elvis* is very limited.

The ‘‘alias’’ program does not work under TOS yet, so the aliases for *elvis* (*ex*, *vi*, *view*, and *input*) are not available.

Currently, character attributes cannot be displayed on the screen. In other words, the ‘‘charattr’’ option does not work very well. It is ironic — the only system that always has a bitmapped display is the only system that does not support multiple fonts!

5.3.12.8. OS9/68k

Elvis was ported to OS9/68k by Peter Reinig.

The Makefile is currently configured to install *elvis* and the related programs in */dd/usr/cmds*. If this this is unacceptable, then you should change the BIN setting to some other directory. Similarly, it expects the source code to reside in */dd/usr/src/elvis*; the ODIR setting is used to control this.

Temporary files are stored in the */dd/tmp* directory. Your */dd/startup* file may need to be modified to prevent it from deleting *elvis’* temporary files.

The program in *alias.c* is linked repeatedly to produce the *vi*, *view*, and *input* aliases for *elvis*. Sadly, the *ex* alias is impossible to implement under OS9, because the shell has a built-in command by that name.

For some purposes, you must give *make* the **−b** option. Specifically, you need this for

```
make -b clean
```

and

```
make -b install
```

5.4. Text Processing

Amoeba does not currently have any text formatters suitable for laser printers of phototypesetters, such as *troff* or TeX. However in the very near future it is expected that TeX will be ported to Amoeba. It does however provide an *nroff* clone which provides a subset nroff. It is described below.

5.5. Nroff

Nroff is a text processor and formatter based on the design provided in *Software Tools* by Kernighan and Plauger. It has been modified to resemble the UNIX *nroff* command. The text and commands found in the file(s) are processed to generate formatted text. Note that one (and only one) of the files can be **–** which reads input from *stdin* at that point. The output always goes to *stdout* which can be redirected by the shell. The **–o** option lets you redirect error output to the specified file rather than *stderr*.

The following command line options are available:

–m*name*	Process macro file *tmac.name*.
–o*file*	Set error log file (default is *stderr*).
–po*n*	Shift output right n spaces (like .po).
–pn*n*	Initial page number (like .pn).
–v	Prints the version information to *stdout*.
+n	Causes output to start with page *n*.
–n	Causes output to stop after page *n*.
–	Input from *stdin*.

Nroff recognizes the following environment variables from the shell. *TMACDIR* is alternate directory to find the files *tmac.** (. for example). The default is */usr/lib/tmac*. *TMPDIR* is an alternate directory to place any temporary files. The default is the current directory.

5.5.1. Commands

Commands typically are distinguished by a period in column one of the input followed by a two character abbreviation for the command function. The abbreviation may then be followed by an optional numeric or character argument. The numeric argument may be an absolute value such as setting the right margin to a particular column, or the argument may be preceded by a plus sign or a minus sign to indicate that the parameter should be modified relative to a previous setting. The following commands are recognized (those marked extension are requests that may be added some day in the distant future).

.ad Begin line adjustment. If fill mode is not on, adjustment is deferred until it is back on. If a type indicator is present, the adjustment type is changed as follows:

Indicator	**Type**
l	adjust left margin only
r	adjust right margin only
c	center
b or n	adjust both margins (default)
absent	unchanged

.af Assign format to number register. The available formats are:

Format	Numbering Sequence
1	0,1,2,3,4,...
001	000,001,002,...
i	0,i,ii,iii,iv,v,...
I	0,I,II,III,IV,V,...
a	0,a,b,...,z,aa,ab,...zz,aaa,...
A	0,A,B,...,Z,AA,AB,...ZZ,AAA,...

The second format above indicates that the field width, i.e., number of digits, is specified by the number of digits in the format type.

.bd Ignored by *nroff*.

.bo Causes the following lines of text to appear in boldface. The optional argument specifies the number of lines to be typed in boldface. Boldface and underlining are mutually exclusive features. The appearance of a boldface command will cause any underlining to cease.

.bp Causes succeeding text to appear at the top of a new page. The optional argument specifies the page number for the new page. The initial value is one and the default value is one more than the previous page number.

.br Causes succeeding text to start on a new line at the current left margin. There is no numeric argument for this command.

.bs Enables or disables the appearance of backspaces in the output text. Underlining and boldface options are implemented by inserting character-backspace-character combinations into the output buffer. This is fine for devices which properly recognize the backspace character. Some printers, however, do not recognize backspaces, so the option is provided to overprint one line buffer with another. The first line buffer is terminated with just a carriage return rather than the carriage return-linefeed combination. A zero argument or no argument to the backspace command removes backspaces from the output. A non-zero argument leaves them in the output. The default is to remove backspaces.

.cc Changes the *nroff* command character to that specified by the character argument. If no argument is provided, the default is a period (.).

.ce Causes the next line of text to appear centered on the output. The optional argument specifies if more than one line is to be centered.

.cs Ignored by *nroff*.

.cu Causes the next line(s) of text to be continuously underlined. Unlike the underline command (see .ul) which underlines only alphanumerics, continuous underlining underlines all printable characters. The optional argument specifies the number of lines of text to underlined. Any normal underlining or boldface commands currently in effect will be terminated.

.c2 Changes the *nroff* no break character to that specified by the character argument. If no argument is provided, the default is a single quote.

.de Causes all text and commands following to be used to define a macro. The definition is terminated by a .en command or the default .. terminator. The first two characters of the argument following the .de command become the name of the new command. It should

be noted that upper and lower case arguments are considered different. Thus, the commands .PP and .pp could define two different macros. Care should be exercised since existing commands may be redefined.

A macro may contain up to ten arguments. In the macro definition, the placement of arguments is designated by the two character sequences, $1, $2, ... $9. When the macro is invoked, each argument of the macro command line is substituted for its corresponding designator in the expansion. The first argument of the macro command is substituted for the $1 in the expansion, the second argument for the $2, and so forth. Arguments are typically strings which do not contain blanks or tabs. If an argument is to contain blanks, then it should be surrounded by either single or double quotes.

.ds Define a string. To initiate the string with a blank or include blanks in the string, start it with a single or double quite. The string can contain other defined strings or number registers as well as normal text. Strings are stored on the macro name space.

.ec Changes the *nroff* escape character to that specified by the character argument. If no argument is provided, the default is a backslash.

.ef Specifies the text for the footer on even numbered pages. The format is the same as for the footer command (see .fo).

.eh Specifies the text for the header on even numbered pages. The format is the same as for the footer command (see .fo).

.en Designates the end of a macro definition.

.eo Turn the escape mechanism off.

.fi Causes the input text to be rearranged or filled to obtain the maximum word count possible between the previously set left and right margins. No argument is expected.

.fl Causes the output buffer to be flushed immediately.

.fo Specifies text to be used for a footer. The footer text contains three strings separated by a delimiter character. The first non-blank character following the command is designated as the delimiter. The first text string is left justified to the current indentation value (specified by .in). The second string is centered between the current indentation value and the current right margin value (specified by .rm). The third string is right justified to the current right margin value. The absence of footer text will result in the footer being printed as one blank line. The presence of the page number character (set by .pc) in the footer text results in the current page number being inserted at that position. Multiple occurrences of the page number character are allowed.

.ft Changes the current font. The choices are R (Times Roman), I (Times Italic), B (Times Bold), S (math special), and P used to request the previous font. P resets the next previous font to be the one just changed, amounting to a swap.

.he Specifies text to be used for a header. The format is the same as for the footer (see .fo).

.in Indents the left margin to the column value specified by the argument. The default left margin is set to zero.

.ju Causes blanks to be inserted between words in a line of output in order to align or justify the right margin. The default is to justify.

.ll Sets the current line length. The default is eighty.

.ls Sets the line spacing to the value specified by the argument. The default is for single spacing.

.lt Set length of three-part titles. Line length and title length are independent. Indents do not apply to titles but page offsets do.

.m1 Specifies the number of lines in the header margin. This is the space from the physical top of page to and including the header text. A value of zero causes the header to not be printed. A value of one causes the header to appear at the physical top of page. Larger argument values cause the appropriate number of blank lines to appear before the header is printed.

.m2 Specifies the number of blank lines to be printed between the header line and the first line of the processed text.

.m3 Specifies the number of blank lines to be printed between the last line of processed text and the footer line.

.m4 Specifies the number of lines in the footer margin. This command affects the footer the same way the .m1 command affects the header.

.na Noadjust. Adjustment is turned off; the right margin is ragged. The adjustment type for .ad is not changed. Output line filling still occurs if fill mode is on.

.ne Specifies a number of lines which should not be broken across a page boundary. If the number of lines remaining on a page is less than the value needed, then a new output page is started.

.nf Specifies that succeeding text should be printed without rearrangement, or with no fill. No argument is expected.

.nj Specifies that no attempt should be made to align or justify the right margin. No argument is expected.

.nr Causes the value of a number register to be set or modified. A total of twenty-six number registers are available designated \na through \nz (either upper or lower case is allowed). When the sequence \nc is embedded in the text, the current value of number register c replaces the sequence, thus, such things as paragraph numbering can be accomplished with relative ease.

.of Specifies the text for the footer on odd numbered pages. The format is the same as the footer command (see .fo).

.oh Specifies the text for the header on odd numbered pages. The format is the same as the footer command (see .fo).

.pc Specifies the page number character to be used in headers and footers. The occurrence of this character in the header or footer text results in the current page number being printed. The default for this character is the percent sign (%).

.pl Specifies the page length or the number of lines per output page. The default is sixty-six.

.pm Print macros. The names and sizes of the macros are printed to *stdout*. This is useful when building a macro package to see how much of the total name space is consumed by the package.

.pn Changes the page number of the current page and all subsequent pages to its argument. If no argument is given, the command is ignored.

.po Specifies a page offset value. This allows the formatted text to be shifted to the right by the number of spaces specified. This feature may also be invoked by a switch on the command line.

.ps Ignored by *nroff*.

.rr Removes a number register.

.so Causes input to be retrieved from the file specified by the command's character string argument. The contents of the new file are inserted into the output stream until an EOF is detected. Processing of the original file is then resumed. Command nesting is allowed.

.sp Specifies a number of blank lines to be output before printing the next line of text.

.ti Temporarily alters the indentation or left margin value for a single succeeding input line.

.tl Specifies text to be used for a page title. The format is the same as for the header (see .he).

.ul Causes the next line(s) of text to be underlined. Unlike the .cu command, this command causes only alphanumerics to be underlined, skipping punctuation and white space. Underline and boldface are mutually exclusive.

The following *nroff* commands, normally available, are currently not implemented in this version:

.fp, .mk, .rt, .vs, .sv, .os, .ns, .rs, .am, .as, .rm, .rn,
.di, .da, .wh, .ch, .dt, .it, .em, .ta, .tc, .lc, .fc, .lg,
.uf, .tr, .nh, .hy, .hc, .hw, .nm, .nn, .if, .ie, .el, .ev,
.rd, .ex, .nx, .pi, .mc, .tm, and .ig.

5.5.2. Escape Sequences

Escape sequences are used to access special characters (such as Greek letters) which may be outside the normal printable ASCII character set. The are also used to toggle certain actions such as font selection. The escape sequences include:

\	backslash character
\e	printable version of escape character
\'	acute accent (equivalent to \(aa)
\`	grave accent (equivalent to \(ga)
\–	minus sign
\.	period
\<sp>	a single, unpaddable space
\0	digit-width space
\&	non-printing zero-width character
\"	beginning of comment
\%	default hyphenation character
\(xx	special character named xx
*x	insert string named *x*
*(xx	insert string named *xx*

\fc	font change (c = R,I,B,S,P)
\nx	interpolate number register x
\t	horizontal tab

5.5.3. Predefined General Number Registers

The following number registers are available for both reading and writing. They are accessed with the \n(xx and \nx escape and can be set with .nr:

%	current page number
dw	current day of the week (1-7)
dy	current day of the month (1-31)
hh	current hours (0-23)
ln	current line number
mm	current minutes (0-59)
mo	current month (1-12)
ss	current seconds (0-59)
yr	last 2 digits of current year

The following number registers are available for reading only:

.$	number of args available in current macro
.A	always 1 in nroff
.H	available horizontal resolution
.T	always 0 in nroff
.V	available vertical resolution
.c	number of lines read from current file
.f	current font (1-4)
.i	current indent
.l	current line length
.o	current page offset
.p	current page length
.v	current vertical spacing

5.5.4. Notes

There are several missing features, notably diversions, traps, and conditionals. This means you cannot use some existing macro packages. There are no **−ms** or **−me** packages as a result. The goal is to (eventually) make *nroff* work with all the SunOS macro packages.

6 Accessing Non-Amoeba Systems

6.1. Remote Login

It is possible to access non-Amoeba systems from Amoeba. For example, it is possible to execute programs on UNIX systems reachable from the Amoeba network. Amoeba provides a special server, called the TCP/IP server, which can be used to send messages to any hosts that can communicate using the TCP/IP network protocol. A special program, *rush*(U) provides a way of executing commands on UNIX machines. (Rush stands for *remote unix shell*.) Several other commands have been provided based on *rush*. For example, *xrlogin*(U) allows you to login to a remote UNIX machine and open an *xterm* window which appears on your Amoeba workstation but which runs on the UNIXmachine. Note that it may be necessary to add the host where the TCP/IP server is running to your *.rhosts* file under UNIX. Otherwise the UNIX hosts may consider that you have insufficient permission to access it. To stop using the remote login you simply log out in the usual way for the UNIX system and the window will go away.

Amoeba's ability to use the TCP/IP network protocol also gives it the possibility of talking over wide-area networks, such as the internet, and to communicate with other systems, including remote Amoeba LANs. This means it is possible to send electronic mail or use FTP. The electronic mail system MMDF is currently being ported to Amoeba which will allow Amoeba to send and receive electronic mail to or from any site on the internet. The changes to MMDF necessary to make it run under Amoeba should be available shortly.

6.2. File Transfer

It is possible to transfer files quickly between Amoeba and UNIX systems without using the *rcp* command of UNIX. The commands to do this are called *tob*(U) and *fromb*(U) *Tob* stands for *to the bullet server* and can send a file from UNIX to the Amoeba Bullet Server. Similarly *fromb* reads a file from the Amoeba Bullet Server and writes it on *stdout* under UNIX. Both these programs run under UNIX and require that the UNIX system have the Amoeba driver built into the UNIX kernel. See the relevant manual pages for more details about how to use these facilities.

6.3. Electronic Mail

At present Amoeba does not provide electronic mail but the mail transport system MMDF (Multi-channel Memo Distribution Facility) is currently being ported and tested. Using the SMTP (Simple Mail Transfer Protocol) on top of TCP/IP it can send mail between Amoeba and any other accessible machines on the internet. It is expected that this will be available in the second half of 1991.

7 Manual Pages

Below is a collection of the manual pages for the utilities and servers in everyday use by users. There will be a brief introduction explaining how to use the man pages, how to get them on-line (if possible) and where to go for more information.

Name

aal − archiver and library maintainer

Synopsis

```
aal {adrtx}[vlc] archive filename ...
```

Description

Aal maintains groups of ACK-object files combined into a single archive file. An index-table is automatically maintained. (A *ranlib*-like mechanism as used under UNIX is unnecessary.) The link editor *led*(U) understands only archives made with *aal*.

The first argument specifies the actions *aal* has to perform. It consists of one character from the set **adrtx**, optionally concatenated with one or more of **vlc**. *Archive* is the archive file. The *filenames* are constituent files in the archive file. The meanings of the characters in the first argument are:

d Delete the named files from the archive file.

a Append the named files to the archive file.

r Replace the named files in the archive file. New files are placed at the end.

t Print a table of contents of the archive file. If no names are given, all files in the archive are listed. If names are given, only those files are listed.

x Extract the named files. If no names are given, all files in the archive are extracted. In neither case does **x** alter the archive file.

v Verbose. Under the verbose option, *aal* gives a file-by-file description of the making of a new archive file from the old archive and the constituent files. When used with **t,** it gives a long listing of all information about the files.

l Local. Normally *aal* places its temporary files in the directory /tmp. This option causes them to be placed in the local directory.

c Create. Normally *aal* will create *afile* when it needs to. The create option suppresses the normal message that is produced when *afile* is created.

Warnings

If the same file is mentioned twice in an argument list, it will be put in the archive twice.

Files

/tmp/ar.* − temporary files

See Also

led(U).

Name

ack − Amsterdam Compiler Kit

Synopsis

```
ack arguments
cc arguments
pc arguments
basic arguments
m2 arguments
machine arguments
```

Description

This program transforms sources in several languages to load files for a variety of machines, internally using several phases. The transformation can be stopped at any phase. Combining sources from several languages is allowed. The run-time system of the first language mentioned, either in the program call name or in the arguments, is automatically included. The libraries of all other languages mentioned, containing most of the run-time systems, are also automatically included. Which combinations of languages and machines are allowed varies in time and depends on the installation.

The actions of *ack* are to repeatedly transform files with a particular suffix into files with another suffix, finally combining the results into a single file.

Different machines can use different suffices, but the following are recognized by most machines:

.b Basic program.

.c C module.

.mod Modula-2 module.

.p Pascal program.

.e EM assembly module in human readable form.

.k Compact EM assembly code.

.m Optimized compact EM assembly code.

.s Machine assembly language code.

.o Object file.

Options

Ack accepts the following flags:

Machine Used

−m*machine*

This flag tells *ack* to generate a load file for *machine*. *Machine* can also be used as the program call name, instead of *ack*. e.g.

```
ack -mm68020 file.c
```

is equivalent to

```
m68020 file.c
```

Output Files

−o Use the next argument as the name of the resulting file. *Ack* produces *a.out* by default. This flag can always be used when *ack* produces a single output file, as in

```
ack -c.s main.c -o new.s
```

The output is produced on *new.s* instead of *main.s*.

−c

−c.*suffix*

Ack tries to transform each source into a file with the *suffix*. When no *suffix* is specified *ack* stops just before the phase where it combines all arguments into a load file, thereby transforming the sources into *.k, .s, .o* or *.m* files. One extra *suffix* is recognized here, *.i*, this tells *ack* to only preprocess all human readable sources, producing files with *suffix .i*.

Note: *ack* refuses to overwrite argument *.e* files.

−t Preserve all intermediate files. If two **−t** are used, *ack* also preserves core dumps and output of failed transformations.

Messages

−w Suppress all warning messages.

−E Produce a complete listing of each Pascal source program. Normally for each error, one message, including the source line number, is given.

−e List only the erroneous lines of each Pascal source program.

−v[*num*]

Verbose. Print information while juggling with files. The optional argument *num* specifies the amount of information printed. The higher *num*, the more information. For example, −v3 specifies that all front and back end invocations are to be displayed.

−fp Use software floating point package.

Preprocessing

−I*dir*

‘‘#include’’ files whose names do not begin with ‘‘/’’ are always sought first in the directory of the *file* argument, then in the directories named in **−I** options, then in directories on a standard list.

−D*name=def*

−D*name*

Define the *name* to the preprocessor, as if by ‘‘#define’’. If no definition is given the *name* is defined as 1.

−U*name*

Remove any initial definition of *name*, before preprocessing.

−P Do not generate line directives.

−C Leave C-comments in.

Debugging

−p This flag tells both the Pascal and C front ends to include code enabling the user to do some monitoring/debugging. Each time a routine is entered the routine *procentry()* is called and just before each return *procexit()* is called. These routines are supplied with one parameter, a pointer to a string containing the name of the routine.

Optimizing

−O

−O*num*

Try to use the optimizers with optimization level <= *num* (default 1). Currently, only the global optimizer has a level > 1. Higher levels will invoke more passes of the global optimizer. The passes which the global optimizer may perform are the following: Inline substitution, Common subexpression elimination, Strength reduction, Use definition analysis, Live variable analysis, Register allocation, Stack pollution, Branch optimization and Cross jumping.

Also, the following flags may be used:

−s*num*

Give an indication to the inline substitution phase, how much bigger the program may get, in percentage. This is only used as a rough indication. The inline substitution phase will not make the program bigger when given −s0.

−a Indicate to the inline substitution phase that it is offered the whole program. This allows it to throw away routines that it has substituted inline.

−Q Give some statistics.

−T Optimize for time.

−S Optimize for size.

In principle, the optimization phases can be run in any order; a phase may even be run more than once. However, the following rules must be obeyed:

- The Live Variable analysis phase (LV) must be run prior to the Register Allocation phase (SA), as SA uses information produced by LV.
- SA should be the last phase.

Also, the following may be of use:

- Inline Substitution (IL) may create new opportunities for most other phases, so it should be run as early as possible.
- Use Definition analysis (UD) may introduce opportunities for LV.
- Strength Reduction (SR) may create opportunities for UD.

The global optimizer is a combiner, so, when using it, offer it all the source files of your program. This is not strictly necessary, but it makes the global optimizer more effective. The current default optimization phases are:

- For −O2: CJ, BO, SP;

- For −O3: CS, SR, CJ, BO, SP, UD, LV, RA;
- For −O4: IL, CS, SR, CJ, BO, SP, UD, LV, RA;
- For higher levels: as for −O4.

−L Disable the generation of code by the front ends to record line number and source file name at run-time.

Libraries

−l*name*
Tells *ack* to insert a library module at this point. For example: the library *mon* contains the routines for systems calls needed by both C and Pascal.

−.*suffix*
When linking multiple *.o* or *.m* files created by separate calls of *ack* together, *ack* cannot deduce the run-time system needed, unless called as *pc*, *basic*, *m2* or *cc*. This flag serves to tell *ack* which run-time system is needed in such a case. For example:

```
ack -c x.c ; ack -.c x.o
```

−r.*suffix*
Most front ends and back ends use one or more run-time libraries. These flags tell *ack* to include the libraries needed when a file with *suffix* would be included in the arguments.

−LIB
This flag tells the peephole optimizer *em_opt* to add information about the visibility of the names used to each output module. This is needed by assembler/linkers when these modules are to be inserted in libraries.

General

−R*program=xxx*
Replace the *program* by the path name *xxx*. The program names referred to later in this manual are allowed here.

−R*program−xxx*
The flag argument *−xxx* is given to *program*.

−R*program:n*
Set the priority of the indicated transformation to *n*. The default priority is 0, setting it to -1 makes it highly unlikely the the phase will be used, setting it to 1 makes it very likely that the phase will be used.

−k Do not stop when an error occurs, but try to transform all other arguments as far as possible.

All arguments without a suffix or with an unrecognized suffix are passed to the loaders, as for flags.

Preprocessor

All C source programs are run through the preprocessor before they are fed to the compiler proper. On larger machines, the compiler has a built in preprocessor. Other human readable sources (Modula-2, Pascal, or Basic programs and machine assembly) are only preprocessed

when they start with a ‘‘#’’.

Ack adds a few macro definitions when it calls the preprocessor. These macro’s contain the word- and pointer-size and the sizes of some basic types used by the Pascal, Basic and/or C compiler. All sizes are in bytes.

Macro	Description
_EM_WSIZE	word size
_EM_SSIZE	size of shorts (C)
_EM_FSIZE	size of floats (C)
_EM_PSIZE	pointer size
_EM_LSIZE	size of longs (C+Pascal)
_EM_DSIZE	size of doubles (C+Pascal)

The name of the *machine* or something like it when the machine name is numeric is also defined (as 1).

The default directories searched for include files differ for each machine.

Programs

Ack uses one or more programs in each phase of the transformation. The table below gives the names *ack* uses for these programs. Internally *ack* maintains a mapping of these names to path names for load files. The table specifies which type of files are accepted by each program as input and the file type produced as output.

Input	Name	Output	Description
.c	cem	.k	C front end
.p	pc	.k	Pascal front end
.b	abc	.k	Basic front end
.mod	m2	.k	Modula-2 front end
.e	encode	.k	Compactify EM assembly code
.k	opt	.m	EM peephole optimizer
.k .m	decode	.e	Produce human readable EM code
.m	ego	.gk	EM global optimizer
.gk	opt2	.g	Second EM peephole optimizer
.m .g	be	.s	Back end
.s	asopt	.so	Target optimizer
.s.so	as	.o	Assembler, relocatable object
.o	led	.out	Linker, Ack object format
.out	cv	a.out	Conversion from Ack to machine binary

Files

/public/module/ack/lib/descr/fe − front end description table

/public/module/ack/lib/`mach`*/descr:* back end description table for *mach*.

Environment Variables

ACKDIR If set, this environment variable overrides *ack*'s

ACKM If set, this environment variable overrides *ack*'s of the default machine it compiles for.

ACKFE If set, this environment variable tells ack where to get the front-end description file.

Diagnostics

The diagnostics are intended to be self−explanatory.

Warnings

Not all warning messages are switched off by the **−w** flag.
Argument assembly files are not preprocessed when fed into the universal assembler/loader.

See Also

basic(U), cc(U), m2(U), oc(U), pc(U).
In the Programming Guide see the reference information on ACK in the *tools* section.

Name

adate − show or set date and time

Synopsis

```
adate [-d] [-s timeserver] [newtime]
```

Description

Without a *newtime* argument, *adate* shows the current date and time (time for short). The output must be interpreted in the user's local time zone.

With a *newtime* argument, it sets the time. The time must be given in GMT as *yymmddhhmm*[.*ss*] (two digits each for year, month, day of month, hours on a 24-hour clock, minutes, optionally followed by a dot and two more digits for seconds). If the *newtime* argument consists of a single minus sign (−), the time is reset to the current time of the default time server, or if the program is run under UNIX, to the current time on UNIX. Setting the time requires special permissions.

Options

−d Show some debugging output which may be useful when debugging the program or a time server.

−s timeserver
This option specifies an alternate time server whose time is shown or changed. The default time server is specified in the file *ampolicy.h* as DEF_TODSVR (typically */profile/cap/todsvr/default*).

Examples

```
adate
```

once printed this:

```
Feb  7 17:18:08 1990
```

To set the time on host bcd to 7 February, 1990, 17:30 hours in time zone MET (which is 16:30 GMT):

```
adate -s /super/hosts/bcd/tod 9002071630
```

This might print:

```
Setting the time...
Time set.  New time: Wed Feb  7 17:30:00 1990
```

See Also

ctime(L), date(U), klok(U), posix(L), stdc(L), zdump(U), zic(U).

Name

ail − Amoeba Interface Language

Synopsis

```
ail [ −?bBcCdgGn ] [ −D name[=val] ] [ −f number ] [ −i inlist ]
    [ −I dir ] [-l language] [ −o dir ] [ −p preprocessor ]
    [ −U name ] [ −v version ] [ + code ]...  [ Source ]
    [ + code ]...
```

Description

Ail is the Amoeba stub-compiler. It reads its input through the C-preprocessor.

Options

−? Print a usage string.

−b Bypass cpp. Useful to avoid *popen()* or replace */lib/cpp*.

−B Rename all files that otherwise would be overwritten as *old-filename*.BAK. Any existing .BAK files that get in the way will be deleted.

−C Stop the macro preprocessor from clobbering comments.

−d This option sets the debug-flag. With this option *ail* reports debugging information on *stdout*.

−D name[=value]
Define the name to the preprocessor, as if by ‘‘#define’’.

−f number
Make sure no output file name is longer than *number*. This is done by deleting the characters just before the first dot.

−g Display a list of generators.

−G Like **−g**, but lists the generators flagged with an underscore. To list all the generators, use **−gG**.

−i inlist
List the names of all read files on file *inlist*. This is done by parsing the #line directives in the output of cpp. Each file name is listed once.

−I dir
Put dir in the path that cpp searches to find #included files.

−l language
Set the default output language.

−n Do not generate output.

−o dir
This flag tells where the output should go by default. If the named directory does not exist, *ail* will create it. Note that if the source uses the ‘‘Output_directory’’, the default is not used.

−p preprocessor

Use a different preprocessor instead of */lib/cpp*.

−U name
Remove any initial definition of a preprocessor symbol.

−v version
Use the capability definition of Amoeba *version*. Legal versions are *3.3* and *3.4*.

+ code
Use *code* as input. This input is not preprocessed.

Warnings

Ail is currently not able to generate code to marshal unions, pointer types, function types, floating point types or void. Enumerated constants must be declared in increasing order, that is, enum { two=2, one=1 } is illegal.

See also

trans(L).

(UNIX ONLY)

Name

ainstall − turn a UNIX a.out file into an Amoeba executable file

Synopsis

```
ainstall [−b extra-bss] [−f filesvr] [−s stacksize]
[−T address] unix-file [amoeba-file]
```

Description

Ainstall is *only* run under UNIX. It creates an Amoeba executable file from loader output produced under UNIX. (It is not needed when running under native Amoeba.) The output file will reside on an Amoeba file server; typically, this is the Bullet Server. The input must be a UNIX file in executable (a.out) format. The output is produced by concatenating an Amoeba-specific data structure, the *process descriptor*, with the unadulterated contents of the UNIX executable file. Offsets for the text and data segments and the symbol table segment in the process descriptor give the offsets relative to the beginning of the file.

This program only works when run on the same architecture as that on which the binary will run. (For example, if you wish to install an Amoeba binary for the mc680x0 then you must run the ainstall program on a UNIX machine that is also an mc680x0.) *Ainstall* should also be compiled with the same *a.out.h* file as the Amoeba binary.

Options

−b bss-extra
: Specifies the number of kilobytes of extra bss (uninitialized data, preset to zero) allocated for the executing program. This is in addition to the bss needed for uninitialized global and static variables declared in the program; the extra space is used by *malloc*. Normally, the bss segment is automatically grown (or a new segment allocated) when *malloc* needs more memory, so it is not necessary to specify this option even for programs with large dynamic memory requirements. The default is 16.

−f filesvr
: Specifies the file server where the output file is created. The default is the standard bullet server, specified by DEF_BULLETSVR in *ampolicy.h* (typically */profile/cap/bulletsvr/bullet*).

−s stacksize
: Specifies the stack size (in kilobytes) for the executing program. Since Amoeba (currently) does not grow the stack automatically, this must be specified for programs that engage in deep recursion or declare large buffers on the stack. The default is zero; in this case a default is chosen by *exec_file*(L) when the cluster is executed (normally 16).

−T address
: Specifies that the text segment must be loaded at the given address (in hexadecimal). This must be specified if the loader was given a **−T** option (with the same address),

since this information is not contained in the *a.out* header. The default is architecture-dependent, and matches the loader default, except that on a vax the default is 400 (hex).

unix-file
: Specifies the input file. This must be in one of the formats documented in *a.out.h*(5). There is no default.

amoeba-file
: Optionally specifies the name of the output file (this is unrelated to the file server used). The default is the same as *unix-file*. If a hyphen (‘‘−’’) is given here, the capability is written to standard output instead of stored at the directory server.

Examples

```
ainstall a.out /home/hello
```

An example of directory-server-independent usage:

```
CAPABILITY=/usr/amoeba/adm/hosts/mark
export CAPABILITY
ainstall -f /bullet a.out - > a.out.cap
```

This leaves the capability on the UNIX file a.out.cap (see also *ax*(U)).

See Also

ax(U), exec_file(L), exec_findhost(L), malloc(L), proc(L).

UNIX man pages: ld(1), a.out(5).

Name

amake − (Amoeba make) stand-alone software configuration manager

```
amake [−f file] [−L dir] [−D var[=value]] [−v[level]] [−t file]
      [−p par] [−r] [−c] [−C] [−k] [−n] [−s] [−w] [cluster] ...
```

Description

Amake is a software configuration manager that was designed to be a tool in the same spirit as *make*, i.e., it is a stand-alone tool that, when run by the user, should invoke precisely those commands that (re-)create a set of target files, according to a description file. The main idea is to make use of previous results as much as possible. Unlike standard *make*, *amake* is also able to exploit possible parallelism between the commands.

In contrast to a lot of ‘‘extended makes,’’ the specification language is not a superset of *make*’s, but a totally new one. For a complete description of the specification language refer to the *Amake User Reference Manual* in the Programming Guide. Here we only will give a short overview, illustrated with an example at the end.

In *amake*, a software configuration can be specified by means of one or more *cluster* definitions. A cluster specifies the *targets* to be constructed from a given set of *sources.* It is *amake*’s task to deduce which *tools* to use, in what order, and how to do it efficiently. The tools available can be defined by the user, but generally a reference to a standard tool library − consisting of a set of *amake* description files − will suffice. In order to be able to decide whether a given file can be used or produced by a certain tool, files are represented as *objects*, each having a set of *attributes* providing pieces of information about the object.

A tool definition, which resembles a function definition found in common programming languages, contains in its header references to the expected *values* of certain attributes, such as ‘type’. Rather than letting the user specify the values of all the necessary attributes, these values can most of the time be derived by *amake* itself, using *attribute derivations*. Rules declaring the possible derivations will generally be present in the library, together with the tools that refer to the attribute values in question, but the user can provide additional ones.

When invoked, *amake* will bring the specified set of clusters up to date by applying only those tools whose inputs have changed since the previous *amake* invocation (if there was any). If an argument (e.g., a compilation flag) is changed between two invocations, *amake* will notice this − because it keeps track of previously executed tools − and will thus re-run the tool with the new arguments. Moreover, old intermediate files produced are retained by default, so changing flags back again will cause it (when all inputs are unchanged) to reinstall the previous outputs, rather than to invoke the tool again. These files are kept in a subdirectory (the *object pool*) residing in the directory from which *amake* is invoked. This subdirectory also contains the *statefile* describing previous tool-invocations.

For a full description of the *amake* language, and *amake* usage in general, refer to the *Amake User Reference Manual*. Another good starting point, if you are interested in the various constructions available, would be to have a look at the standard tool library.

Options

The options available are the following:

−f *file*

The default specification file *amake* reads is *Amakefile* in the context in which it is invoked. A specification file with a different name can be specified by this option. Only one **−f** option may be specified.

−L *dir*

Files having relative path names, that are to be %included are normally looked for in the default system directory (see the Files section) and the current working directory. Invoking *amake* with **−L** options causes it to look in the specified directories first, in the order in which the options are supplied.

−p *par*

By default *amake* runs at most 4 commands in parallel, in order to prevent the local machine from thrashing. A different number (currently in the range of 1 to 20) can be specified with this option. When the local machine has little memory, a lower number might give better results, while a higher number will be appropriate when several remote machines or (under Amoeba) processors are available. Supplying 0 as argument to this option causes *make*-like behavior; only one command is started, and *amake* waits for that command before it continues. (When **-p 1** is given, one command is running on the *background*, and *amake* does some useful work in the meantime).

−r

Under UNIX, this option causes remote execution on a pool of workstations. A default pool is compiled into *amake*, but a different one can be specified in the environment variable `RSH_POOL`. Note that when using this option you had better be sure that all the machines in the pool are up, and also (with NFS) that your *$HOME/.rhosts* contains the machine on which you are running *amake*. Furthermore, each machine has got to have the same file structure, i.e., if on the machine running *amake* a file is reachable under the name */usr/users/versto/src/monkee.c*, the same should be the case for each of the other machines in the pool.

−D*var*[=*expression*]

An *amake* variable can be set on the command line by means of a **−D** option. The value is optional (default %true), and is to be specified in *amake* syntax. Take care to quote the expression properly, if it contains characters with a special interpretation in the shell you are running. Also, the argument required by the **−D** option has to be one string, so

```
amake -D src = hello.c
```

will not be interpreted the way you expect. In the Bourne shell, specify

```
amake -D 'src = hello.c'
```

(i.e., explicitly made one string) or

```
amake -Dsrc=hello.c
```

(i.e., without spaces) instead.

−k

By default intermediate files, generated while creating the targets specified, are kept

away from the user's eyes, and moved to and from the object pool *amake* keeps, whenever necessary. The **−k** option leaves intermediate files in the current working directory. This may be used, for instance, when they contain information the user needs for debugging purposes.

−n

Stops *amake* from actually executing commands for targets that need updating. The commands are only written on standard output. It is advisable to use this option (possibly in combination with the **−v** flag, see below) when a new *Amakefile* is used for a certain configuration.

−v[*level*]

Causes *amake* to be more verbose than usual. This option might help to find out why it does not do what the user intends. Although *amake* is relatively intelligent, it is not clairvoyant. The verbosity wanted can be specified as parameter to this option: the higher the more verbose. The default verbosity level is 1. In this case, *amake* reports, among other things, the reason why it considers a tool not to be applicable within some cluster.

−t *file*

Causes *amake* to consider files having the same last component as the argument of this option to be updated (or ''touched,'' in UNIX terms). This can be used when a command executed in a previous run did not deliver a non-zero exit status, even though it failed. Tools having a touched input will be re-invoked by *amake*. For example,

```
amake -t source.c
```

will force a recompilation of files called source.c, while

```
amake -t source.o
```

will only force re-invocation of the loader.

−w

Warnings about questionable constructions, unsuspected situations, etc., are not given when this option is supplied.

−s

By default *amake* prints the commands it is executing on standard output. Specifying this option avoids this.

−c

This option should be specified if you do not want to use *amake*'s feature of caching objects stemming from previous tool invocations with different options. Generated files not needed in the current run will be removed from the cache. You might want to use this option when you are short on disk space. You can also use it as a kind of 'garbage collection', e.g., when you have once created a target with some exotic option you will not be needing again, in the near future. Ideally, *amake* ought to have some clever aging algorithm to do this automatically (and it probably will, one day.)

−C

This a more drastic version of the **−c** option; it removes the entire *amake* object pool, statefile and targets. You can use this when your software configuration is ready, and its targets installed. When an *amake* bug or some external cause has damaged the

statefile containing the *amake* administration, this will not work. You will then have to give a command

```
rm -rf .Amake target ...
```

yourself.

cluster...

All trailing arguments are taken as names of clusters or targets that are supposed to be updated. By default the first cluster encountered is assumed to require updating, together with the clusters producing sources for it.

Diagnostics

Commands are reported on standard output as they are executed, prefixed by the machine where the execution takes place. When all commands are executed locally, they are prefixed with a virtual processor name. The diagnostics of a command are buffered, i.e., redirected to temporary files and shown when the command has finished.

Other diagnostics, warnings and error messages produced by *amake* itself, are intended to be self-explanatory. Not all of them are but they will be described once things have settled down.

Environment Variables

The environment variable `RSH_POOL` can be used to specify the machines available for remote execution of commands (see the **−r** option). The names of the machines are to be separated by white space.

The *amake* language contains %import directives, which can be used to set *amake* variables according to the environment *amake* is started from. Likewise, %export directives are available to put *amake* variables in the environment supplied to commands invoked by *amake*.

Files

Amakefile	Default name of the configuration description file.
.Amake	A subdirectory in the working directory from which *amake* is called. It contains files (possibly several versions of some) resulting from previous tool invocations.
.Amake/Statefile	Contains a description of invocations made in previous builds.
/usr/local/lib/amake	Contains a set of files defining tools needed for software development in C, including the parser generators yacc and LLgen. All you should have to do is to put `%include std-amake.amk` at the top of your *Amakefile*.

Warnings

As *amake* decides where and how it starts a command, it is capable of deciding whether an argument needs quotation. (This may be the case when a shell is used.) If, for example, the C-compiler is to be called with an option defining a string, it should be specified with

```
CFLAGS = '-DSTR="string"';
```

When, as is currently the case under UNIX, the Bourne Shell is used to start the C-compiler, *amake* will insert back quotes before the " characters in the command.

The fact that there is only one object pool per context allows previous invocations to be shared between different *Amakefile*s. It is not advisable, however, to do this for specification files that have no sources in common. In that case a large statefile has to be read (and possibly written) at each *amake* invocation, resulting in poorer performance.

Amake is known to use quite a lot of memory when updating a large configuration, or when it has to maintain many instances of the same configuration. In the latter case, it helps to invoke *amake* with the **−c** flag (described above) once in a while.

Targets created with *amake* are hard-linked into the cache. Therefore be sure to copy (rather than move) a target when you install it. Otherwise a future *amake* invocation may cause an inconsistent version to overwrite the installed one.

The remote execution option should only be used to update a working configuration, because we (only recently) found out that the rsh command fails to deliver a bad exit status when a remote command fails. We have concrete plans for a solution to this problem, which is more efficient than the current implementation as well.

Example

The following shows what the specification of simple software configuration might look like in *amake*.

```
%include std-amake.amk;

%cluster
{
   %targets comp[type = program];
   %sources parse.y, scan.l, comp.c, defs.h, comp.a;
   %use     cc-c(flags => '-DAMOEBA');
};
```

The %include directive on the first line causes *amake* to read the file *std−amake.amk*, which in turn lets it %include a standard set of tool definitions and derivation rules. The type-attribute of the target "comp", defined in line 5, is added explicitly and forces *amake* to apply a loader as last step in the construction of targets. The sources, introduced on line 6, have type *yacc-src*, *lex-src*, *C-src*, *C-incl* and *library* respectively, all of which can be derived using analysis of the suffixes. Line 7 shows that we can influence tool-behavior (in this case of the C-compiler) by supplying other values for parameters than the default ones. Alternatively, an assignment to the global variable CFLAGS could be made, which (like in *make*) is used to alter the default argument globally.

See Also

In the Programming Guide
Amake User Reference Manual.

Name

aps − show process status

Synopsis

```
aps [-a] [-u username] [-l number] [-v] [directory] ...
aps [-a] [-u username] [-l number] [-v] -m [machine] ...
```

Description

The first form prints information about the processes running on each host found in each directory. If no directories are given, the default pool-processor directories are used. These processors usually contain all the processes of an ordinary user.

The second form (with **−m**) causes the remaining arguments to be interpreted as machine names instead of directories. Information about the processes of each specified machine is printed. If no machine names are given, all known machines are used.

In either form, by default, only the processes of the current user are listed (but see **−a**, below).

Normal output is columnar, and consists of a header containing column labels, followed by one line of information for each selected process. The displayed information includes, by default, the machine name (HOST), the Amoeba process table entry number (PS — the name of the entry under the *ps* directory of the machine), the status (ST — R=running, S=stopped), and the command that was used to invoke the process (COMMAND).

Note that if the HOST column shows *x* and the PS column shows *n*, the process capability can be accessed as *hostdir/x/ps/n*, where *hostdir* is the directory containing all the host machines of the local network (usually */public/hosts*). Among other uses, this can be supplied as an argument to *stun*(U) or *std_info*(U).

Note also that the command information is extracted from the string returned by a std_info call to the process's capability, and is not guaranteed to be correct.

If certain information is not available, an error indication is given in parentheses.

Options

−a Show all processes on the selected machines, not just the ones belonging to the current user. If specified, a USER column is added to the output, showing the user that invoked each process. This is obtained from the std_info transaction to the process capability or to the process owner capability, and is not guaranteed to be correct.

−u *username*
Show only the processes belonging to the specified user. This option is ignored if **−a** is specified.

−l *n* Instead of printing machine name in the first column of each process line, precede the listing for each machine by a syntactically distinguished header line showing the machine name and the load on the machine (as obtained from ppload). Pad the output for each machine with blank lines to at least **n** lines, to make the output of repeated calls appear at the same height. This is useful when the output is being repeatedly

piped to a screen update program, e.g., by typing

```
!Gaps -al 8
```

repeatedly to *vi*.

−v Add columns of information about the process that owns each listed process, obtained by doing a std_info transaction to the owner capability. The OWNER column is a string identifying the owner program (usually the session server or an *ax* server). If the process is using UNIX-emulation, the UNIX PID is a unique identifier for the process, otherwise the column is blank. The UNIX PID can be supplied to the *kill*(U) command to signal the process.

If **−l** is in effect, **−v** also causes an extra line to be added to each machine header, listing the version string returned by the machine.

Examples

The command

```
aps -av -m vmesc2 vmesc4
```

shows all processes on machines *vmesc2* and *vmesc4*:

```
           UNIX S
HOST    PS  PID T USER    OWNER              COMMAND
vmesc2   2      R condict ax server          session -a
vmesc2   4      R mjh     ax server          session -a
vmesc2   3   18 R mjh     session server     artserv -d
vmesc2   5   22 R mjh     session server     artcli
vmesc4   1      R condict ax server          run /super/cap/runsvr/
vmesc4   3   39 R sater   session server     -sh
vmesc4   5    2 R mjh     session server     -sh
vmesc4   7   19 R mjh     session server     vi temp
```

The command

```
aps -al 7 -m vmesc2 vmesc4
```

shows the same processes, but without the owner information and using the fixed screen format:

```
----- vmesc2: 1.526 of 1.976 mips, 1216 kb free, up 17 hrs 19 min -------------
  2  R condict session -a
  4  R mjh     session -a
  3  R mjh     artserv -d
  5  R mjh     artcli

----- vmesc4: 1.378 of 1.976 mips, 1856 kb free, up 23 hrs 14 min -------------
  1  R condict run /super/cap/runsvr/
  3  R sater   -sh
  5  R mjh     -sh
  7  R mjh     vi temp
```

See Also

kill(U), kstat(A), ppload(U), stun(U).

Name

ascii − strip all the pure ASCII lines from a file

Synopsis

```
ascii [-n] [file]
```

Description

Sometimes a file contains some non-ASCII characters that are in the way. This program allows the lines containing only ASCII characters to be grepped from the file. With the **−n** flag, the non-ASCII lines are extracted. No matter whether the flag is used or not, the program returns an exit status of true if the file is pure ASCII, and false otherwise.

Examples

```
ascii file > outf
```

Write all the ASCII lines from *file* on *outf*.

```
ascii -n < file > outf
```

Write all the non-ASCII lines from *file* on *outf*.

Name

aterm − X11-stdwin terminal emulator

Synopsis

```
aterm [ options ] [ program [ argument ... ] ]
```

Description

Aterm creates a window on an X11 server in which it runs an interactive Amoeba application, normally the shell. Characters typed into the window are read by the application from standard input, and characters written by the application to standard output (or to the standard error stream) appear in the window. Programs using the *tios*(L) interface or the Posix *tc_getattr* and *tc_setattr* functions (see *posix*(L)) will believe that an input or output capability or file descriptor connected to *aterm* is a tty, and behave accordingly (e.g., the stdio library will turn off output buffering).

Aterm is a so-called *terminal emulator program*, which means that it interprets certain escape sequences in the output as commands to move the cursor around, to insert or delete lines or characters, to write underlined and inverse video characters, etc. It understands most escape sequences defined by the relevant ANSI standard (most of which are also DEC VT-100 escape sequences). This is used by editors like *be*(U) and *elvis*(U), and by other programs that use the *termcap*(L) library.

Options

Aterm is an X11 STDWIN application, which means that it accepts all standard options recognized by such applications. (see X(U) and X11stdwin(U)) Particularly relevant are:

−**display** *display-name*
: specifies the display to be used, in the format *host***:***display*[**.***screen*], e.g., `tjiftjaf:0.1` refers to display 0, screen 1 of host *tjiftjaf*. The host name may be replaced by an IP address (four decimal numbers separated by dots; to find a host's IP address, try grepping for its hostname in */etc/hosts.)*

−**geometry** *geometry-string*
: specifies the initial size and/or position of the window, in the format [*width***x***height*][+*x*+*y*], e.g., `600x400+100+100`. Note that it is normally best to specify a position only, e.g., `+100+100`, since the size of the window is easier specified with **−c** and **−r** described below.

−**reverse** makes the terminal use white characters on a black background.

Other options (specific to *aterm*):

−**c** *columns* Sets the number of columns of the emulated terminal. The default is 80.

−**r** *rows* Sets the number of rows of the emulated terminal. Default 24.

−**i** Prefix the name of the executed command in its argv[0] with a hyphen. This makes the shell read your profile.

−**j** Turn on jump scrolling. This speeds up scrolling by combining multiple scroll operations into one where possible. This does not look very nice, but may be the only way to get a reasonable performance on slow displays.

−N Do not start a command. This should be used with **−p** or **−P** below.

−p *pathname* Specifies a path name where to publish the capability.

−P *pathname* Similar to **−p**, but a two digit hex number is appended to the path to make it unique. Conventional use: **−P** /dev/tty.

−R Turn on auto resizing: when the window is resized, the size of the emulated terminal changes to match. Normally, the emulated terminal keeps the size specified initially, and the scroll bars can be used to view parts of it that are not visible in the window.

−s *lines* Sets the number of lines in *scrollback* memory (default 100).

−T *title* Sets the title displayed in the title bar (if your window manager supports title bars).

program [*argument* ...]
If any non-option arguments are given, they specify a command to be run in the window. The default is */bin/sh*.

Many of these options can be changed while the program is running, either using a menu or by changing the window's geometry. Only the X11 STDWIN options can be specified as X11 resources.

Diagnostics

When an illegal option is specified, a usage message is printed which also shows the defaults for the options.

The most annoying complaint is

```
fatal error: winitnew: cannot open display (display-name)
```

which is issued for almost any problem while trying to connect to the server. Most problems occur when trying to connect to a UNIX host via TCP/IP (see *tcpip*(A)). Possible causes include (but are not limited to):

- Illegal syntax for *display-name* (e.g., the colon is omitted);
- The host name is not found in */etc/hosts*;
- You have specified a non-functioning TCP/IP server in the TCPIP_SERVER string environment variable;
- TCPIP_SERVER is not set and there is no TCP/IP server listening to */public/cap/tcpipsvr/any*;
- The TCP/IP server is overloaded;
- The X server host does not allow the host where the TCP/IP server runs to connect. This is accompanied by the message

  ```
  Client is not authorized to connect to Server
  ```

  and is remedied by proper use of *xhost* (see *X(U))* on the X server machine − use either

  ```
  xhost +
  ```

  to allow all hosts to connect, or

```
xhost + hostname
```

to allow just the host where the TCP/IP server runs to connect.

Environment Variables

Like all X11 applications, *aterm* uses the string environment variable TCPIP_SERVER to find a TCP/IP server when asked to connect to a TCP/IP host; if this variable is not set, */public/cap/tcpipsvr/any* is used.

The string and capability environments of the executed command are inherited from *aterm*, with some changes: In the string environment, `TERM` is set to ‘‘aterm’’ and `DISPLAY` is set to the **−display** argument if given. In the capability environment, `STDIN`, `STDOUT`, `STDERR` and `TTY` are set to *aterm's* capability.

Warnings

A section describing the menus and general interaction with *aterm* is needed.

A section describing the recognized ANSI escape sequences is needed.

Because X11 STDWIN initialization is performed before option parsing, the usage message can only be printed if a successful connection to a server can be made. As a work-around, *aterm* recognizes the option **−?** before initializing if it is the first option.

Except for the arrow keys, X11 STDWIN does not support any special keys on the keyboard, such as function keys or the numeric pad.

The default command */bin/sh* will not work when there is no session server or when there is no capability for the session server in the capability environment (see *session*(U)). You can use

```
ax /bin/aterm /bin/session /bin/sh
```

to let the session server start a shell, or attach to a running session server with

```
ax /bin/aterm
```

from UNIX.

Cutting from an *aterm* window and then pasting it in an *xterm* window hardly ever works, since *aterm* only uses the cut buffer while *xterm* (by default) uses the primary selection if possible.

Scrolling from a place which is covered by another window to a visible spot sometimes leaves a cleared region. Refreshing the screen helps.

Examples

```
aterm -i -P/dev/tty
```

This starts a shell in a window, and publishes the tty capability as */dev/ttynn* where *nn* is a hexadecimal number. The shell will read your `.profile`.

```
aterm /bin/elvis myfile.c
```

This starts *elvis* in a window. The window disappears soon as you exit *vi*. Note that you must supply a complete path for the program.

See Also

allo(A), ax(U), be(U), posix(L), session(U), tios(L), termcap(L), tcpip(A), X(U).

Name

ax − execute program using specified environment and host

Synopsis

```
ax [options] program [argument] ...
```

Description

Ax starts a new Amoeba process from Amoeba (or from UNIX, given an executable Amoeba file written by *ainstall*(U)) When invoked under Amoeba it is either to choose a specific processor on which to run a job or to specify a special environment. Normally the shell will start jobs for you.

From UNIX *ax*is When invoked under UNIX, *ax* acts as an I/O server, once the process is executing, to handle standard input and output from the process. Two copies of the I/O server are forked, to allow simultaneous input and output. In this state, interrupt and quit signals are propagated to the process (causing its termination) or to the session server (which may pass it on to the appropriate process group). Note that the Amoeba process will not be able to communicate with the outside world, if the *ax* job is killed or dies.

When invoked under Amoeba, the new process inherits the `TTY`, `STDIN`, `STDOUT` and `STDERR` capabilities from *ax*, so no I/O server is necessary. In this case, *ax* exits immediately after startup, after first arranging that the new process thinks its owner is the owner of the *ax* process, and that the owner of *ax* thinks the new process is its child . This causes the shell to wait for the process started by *ax,* instead of terminating its wait when *ax* exits.

Options

−c Write the capability for the running process to standard output. In this case, all further output (e.g., the process' standard output) is redirected to the standard error output of *ax*.

−d Print debugging output.

−o owner
Sets the capability for the process owner. The owner receives a checkpoint transaction when the process exits, is stunned or causes an exception. By default, *ax* itself acts as an owner, printing a register dump and terminating the process when it is contacted. The register dump is printed using one of the *pdump*(U) family of programs.

−s stacksize
Sets the stack size in kilobytes. The default is 16.

−m host
Sets the host on which the process is executed. This may either be a host name or a full path name giving a host capability, as accepted by *host_lookup*(L). By default a suitable pool processor is chosen by *exec_findhost*(L).

−p procsvr
Sets the capability of the process server. This is really a different way of overriding the host on which the process is executed; it is more precise because any capability can be

specified, not just a host name.

−C name=capability
: Adds an item to the capability environment (accessible through *getcap*(L)). By default the capability environment contains the following items: `STDIN`, `STDOUT`, `STDERR`: capabilities for the process' standard input, output and diagnostic output streams (implemented by *ax* acting as an I/O server); `HOME`, `WORK`: capabilities for the process' home and work directory; set to the home and working directory of the UNIX user executing *ax*. `_SESSION`: capability for the session server, taken from /dev/session, if it exists. The **−C** option can be used to override these standard items as well as to add new items.

−D corefile
: Sets the name of the file *ax* dumps the process descriptor on in case the process dies of an exception. This option does not work when *ax* runs under the session server, because it takes over the ownership of the process.

−E 0
: Initialize the environment to empty. If this option is not given, the environment that *ax* inherits is passed on to the executed program. In all cases, two items are forced to standard values: `HOME` is set to / and `SHELL` is set to */bin/sh*. These, too, can be overridden using the **−E** *name=string* form below.

−E name=string
: Adds an item to the string environment, overriding any definition of the same name inherited from the shell or set by default.

−1 Start only a single I/O server. This is slightly more efficient if the process produces very little output, but can cause delays if it produces much, and makes it impossible to perform output while also waiting for input (e.g., in a different thread). By default, two I/O servers are started.

−n Don't start any I/O servers. *Ax* will exit as soon as the process is started. This is useful to start long-running servers with output redirected to some logfile using the **−C** option.

−x pathname
: Specifies the Amoeba path name for the program. By default the program name is used. If a hyphen (''−'') is specified, a capability is read from standard input.

program
: Specifies the name of the program (passed in argv[0]). Unless the **−x** option is used, this is also the Amoeba path name of the executable file.

[argument] ...
: Optional program arguments (accessible through the normal *argc/argv* mechanism).

If you have an Amoeba directory called */process*, *ax* stores the capability for the running process there, and prints the name under which it is stored. This capability is removed when *ax* notes that the process is terminated. The name is the last component of the executable, with an optional number suffix to avoid name conflicts. It may occasionally be necessary to remove invalid capabilities from this directory; if more than 10 capabilities for the same program exist with different suffixes, new capabilities are not stored.

Environment Variables

The `BULLET` string environment variable can be set to override the name of the file server used to create the stack segment for the process (which also contains the environments and arguments).

Examples

To run the binary in */home/hello* on host *bcd*:

```
ax -m bcd /home/hello
```

An example of directory-server-independent usage, assuming a capability for an executable file can be found on the UNIX file *a.out.cap* (see also *ainstall* (U)):

```
CAPABILITY=/usr/amoeba/adm/hosts/mark
BULLET=/bullet
export CAPABILITY BULLET
ax -m / -x - hello < a.out.cap
```

See Also

ainstall(U), exec_file(L), exec_findhost(L), getcap(L), getenv(L), host_lookup(L), pdump(U).

Name

banner − print a banner

Synopsis

```
banner arg ...
```

Description

Banner prints its arguments on *stdout* using a matrix of 6 x 6 pixels per character. The ‘‘@’’ sign is used for the pixels.

Example

```
banner happy birthday
```

prints a banner saying ‘‘happy birthday.’’

Name

basename − strip off file prefixes and suffixes

Synopsis

```
basename file [suffix]
```

Description

The initial directory names (if any) are removed yielding the name of the file itself. If a second argument is present, it is interpreted as a suffix and is also stripped, if present. This program is primarily used in shell scripts.

Examples

```
basename /user/ast/file
```

strips path to yield *file*.

```
basename /user/file.c .c
```

strips path and *.c* to yield *file*.

Name

bawk − pattern matching language

Synopsis

```
bawk rules [file] ...
```

Description

Awk is a pattern matching language. *Bawk* is Basic Awk, a subset of the original. The language is described in the *tools* section of the *Programming Guide*. The file name ‘‘−’’ can be used to designate *stdin*.

Examples

```
bawk rules input
```

Process *input* according to *rules*.

```
bawk rules - > out
```

Input from terminal, output to *out*.

Name

be − screen-oriented text editor

Synopsis

```
be [−b bindings-file] [−m margin] [−t tag] [+[start]] [file] ...
```

Where `+start` can be

```
+
+$
+number
+/pattern
```

Description

Be is a general text editor, intended to be easy to learn and easy to use for novices, yet fast and powerful for experts. It borrows ideas from several sources, notably Emacs, the Apple Macintosh, and the programming environment for the ABC language developed at CWI.

This manual page does not contain the information needed to use *be*; if you happen to have entered it, you can get out immediately with ^C, or save a file you have managed to create with ^X-^F. Emacs users will find that (if the default key bindings are in effect) most of the simpler Emacs commands work. Also, the [help] command, which in fact just displays the key bindings for selected commands, is by default bound to ESC-?, so you can explore the available commands yourself if you have to (type a carriage return in response to the ‘‘help search string’’ prompt to show all the help there is).

Options

−b bindings-file
: Read additional key bindings from *bindings-file*. The format of a bindings file is given below. More than one **−b** option may be given and will be processed in order.

−m margin
: Sets the right margin. This is a number specifying that lines longer than *margin* characters are broken when typed in. A value of zero disables the feature (which is the default) but retains the last non-zero value given as the default for the command [right-margin-set]. Initially this value is set to 72.

−t tag
: Performs an initial [goto-tag] command with the given tag. Tags can be abbreviated and the tags file need not be sorted, but otherwise this works the same as the **−t** option to *vi*.

+[start]
: This option must appear at most once as the last option before the files, and is again almost the same as the same option to *vi*. It applies to all files given on the command line. It can have three forms: If only a + or +$ is given, the editor starts with the focus at the end of the file. If +number is given, the editor starts with the focus at that line number. If +/pattern is given, the editor starts with an initial regular expression search for the given pattern.

file ...
: Any files named on the command line are edited, each in their own window. If no files are given and no file is edited as a result of the **−t** option, an initial [list-directory] command is issued.

Environment Variables

Be uses the following (string) environment variables.

`TERM`
: terminal type.

`TERMCAP`
: substitute for /etc/termcap or termcap entry.

Files

Be uses the following file names.

,*
: When a modified file is first written back, the original is retained until the editor is left with a comma prefixed to its name.

tags
: contains the database for the [goto-tag] command.

*/usr/tmp/benv**
: contain back-ups.

*.../lib/benv_**
: terminal-specific key bindings (suffix used is `$TERM`).

.../lib/benv
: executable binary of the editor.

.../lib/be.recover
: recovery shell script.

/etc/termcap
: default terminal capabilities database.

Warnings

The [goto-cursor-position] command needs the addition of two non-standard items (sp and cp) to the termcap entry; it will issue the message ‘‘cursor sense failed’’ if either is not found. (Of course, this feature does not work on most terminals anyway − the only ones known to work are HP’s and locally hacked DMD’s, as well as *xterm* and *aterm* with hacked termcap entries.)

Examples

To edit the file */etc/passwd* with the focus initially at the end:

```
be + /etc/passwd
```

To edit the function foobar (assuming the tags file has been created correctly):

```
be +/foobar
```

Bindings File Format

Blank lines and lines starting with # are treated as comments and are ignored. The format of all other lines is *name=string* where *name* is (any unambiguous abbreviation of) a command name, and *string* is the new binding for the command. No extra blanks should appear on the line.

In the binding, the following string escape sequences are recognized as in C: \n, \r, \b, \t, \f, \ddd (one to three octal digits). Also recognized are \e (for ESC) and ^X (for meaning CTRL−X, for any character X).

Old bindings of a command are not removed by rebinding it, but only when a new command is bound to them. To explicitly unbind a command, use the command name ‘‘ignore’’.

For a list of command names, try the [help] command (ESC-? followed by CR). The contents of the help window is a syntactically correct bindings file.

See Also

ed(U), elvis(U).

In the Editors section of the User Guide the section entitled *The BE Editor*.

Name

brol − Backup and restore tool

Synopsis

```
brol -o [flags] ... dumpfile path ...
brol -[it] [flags] ... [dumpfile [path] ...]
brol -g [flags] ... path ...
brol -[cl] [flags] ... source destination
```

Where flags can be

```
-v
-x path
-X pattern
```

Description

Brol can save an entire Soap directory graph to an archive file from which it can later restore that same graph. The archive is usually written to and read from a cartridge using *tape*(U). It also performs related functions such as listing the objects in a directory graph, listing the contents of an archive, and copying a directory graph to another root. There are options to skip certain directories or file names matching a given pattern.

A simple tree traversal algorithm such as is typically used by the utilities *tar*(U) and *find*(U) may get into an infinite loop under Amoeba; it is necessary to remember at least the capabilities of all directories encountered on the path to a directory, to avoid going around in circles. *Brol* in fact remembers the capabilities of all objects it has encountered, so it will not only avoid loops but in general dump each object at most once.

Options

Exactly one mode option must be specified, chosen from the list below. The interpretation of path name arguments depends on the mode. Whenever a path name specifies an archive file, a minus sign (−) may be substituted, meaning that standard input or output is read or written. If no path name needs to follow the archive file name argument, the archive file name may be omitted with the same effect. Mode options are:

−o Output mode. The first path name argument specifies the archive file. At least one more path name must follow specifying the starting point(s) of the directory graph(s) to be written to the archive file.

−i Input mode. The first path name argument specifies the archive file. Following path names specify starting points of subsets of objects to be restored; if no subsets are specified, the entire archive is restored.

−t Tabulation mode. This lists the contents of the archive rather than restoring from it; it is otherwise similar to input mode.

−g Name generation mode. This does not manipulate archives; it traverses one or more directory graphs whose starting point(s) are given as arguments and prints the names of the objects found. These are exactly the objects that would be written to the archive in output mode.

−c Copy mode. This requires exactly two path name arguments, a source and a destination: the directory graph starting at the source is copied to the destination.

−l Link mode. This is like copy but avoids copying bullet files, to save disk space; instead, their capabilities are copied, creating links. Directories are always copied, and non-directory, non-bullet objects are always linked. (The danger of links is that when the original is destroyed, the link is also unusable; however, because bullet files are immutable, other changes to the original will not affect the link.)

The remaining options may be combined with any of the above:

−v Verbose. This may be repeated to increases the amount of output generated. There are four levels of verbosity: silent, names only, names and status information, and extended status information. Tabulation and name generation mode (**−t** and **−g**) start at the second level (names only); the other modes are silent by default (except for error messages).

−x path
Specifies a path name which should not be dumped/restored/listed/etc. (as appropriate for each mode). This may be repeated to specify multiple exceptions. If a directory is specified in this way, the subgraph started at this point will not be traversed at all. Other names for the same object will be treated as links (thus, the object itself will not be copied to the archive at all). In input mode, objects will not be restored when an initial section of their path name matches the exception path.

−X pattern
Specifies a file name matching pattern with a syntax (hopefully) identical to that of the shell; objects whose last path name component matches the pattern are skipped. If a directory is skipped the subgraph starting at that point will not be traversed. Unlike the **−x** option, objects that are skipped through this option may still be reached by other names. This option, too, may be repeated to specify multiple exceptions.

Informative Output Format

Output mode writes its information on *stderr* (since *stdout* may be used as the archive file); the other modes write to *stdout*. As explained with the **−v** option, the amount of informative output can vary. The ‘‘names only’’ output contains just the path names of the objects, one per line. The ‘‘names and status’’ output contains the following items for each object:

(1) Up to four (really SP_MAXCOLUMNS) columns of hex numbers giving the column masks for the entry. This may be empty for the starting points (and for /, where the notion of column masks is meaningless).

(2) A parenthesized hex number giving the rights actually found on the object’s capability reached through this path.

(3) The object type: B for Bullet file, C for Capability (of unknown type), D for Directory, L for Link to previously encountered object.

(4) Per-type information: number of bytes for bullet files, number of rows and columns for directories, blank for other types.

(5) The object’s path name.

(6) For links only, an arrow (`->`) followed by the name of the object to which the link points.

‘‘Extended status’’ adds a second line for directories containing the indented text ‘‘(Columns: *name* ...)’’.

Warnings

Due to a missing interface in the Soap Server, it is not possible to restore objects’ modification times from archives (the time is put on the archive however, so if that interface is ever added...).

Copy mode is currently not implemented; use link mode instead.

Input mode refuses to overwrite existing files. There should be at least an option to allow this.

Examples

To list the file names of everything in directory *src* and below:

```
brol -g src
```

This might print:

```
src
src/main.c
src/test
src/a.out
src/test/testinput
```

To dump the contents of directory *src* to file *src.brol:*

```
brol -o src.brol src
```

To get a verbose listing of the contents of file src.brol:

```
brol -t -v src.brol
```

This might print:

```
             (ff) D       3 cols  src
ff ff ff     (ff) B    2334 bytes src/main.c
ff ff ff     (ff) D       3 cols  src/test
ff ff ff     (ff) B   59126 bytes src/a.out
ff ff ff     (ff) B    3574 bytes src/test/testinput
```

To restore everything from src.brol except the directory src/test, and get a listing of the files restored:

```
brol -i -v -x src/test src.brol
```

This might print:

```
src
src/main.c
src/test
src/a.out
src/test/testinput
```

To make a backup copy of directory *src* in *src.bak*:

```
brol -c src src.bak
```

File Format

The archive file format is unique to *brol*; standard archive formats such as *tar* (which are also available in Amoeba) do not provide space for the information about files and directories that must be saved in order to be able to restore them to identical state (such as directory column names, column masks and capability sets). The exact format is easily gleaned from the source code; it should be sufficient here to state that headers are 512 bytes, file data is padded with zero bytes to multiples of 512 bytes, and headers are byte-order independent. The program prints an error and aborts when a header cannot be parsed or an I/O error occurs on the archive file.

See Also

dirgraph(L), find(U), tape(U), tar(U).

Name

bsize − report the size of a bullet file

Synopsis

```
bsize file [file2 ...]
```

Description

For each file name on the command line *bsize* prints the name of the file followed by a colon followed by a space, followed by the size of the file in bytes, followed by a newline.

Example

```
bsize /bin/cat /bin/echo
```

prints

```
/bin/cat: 49376
/bin/echo: 41184
```

Name

a2c, c2a − Convert capabilities to ar-format and vice versa.

Synopsis

```
a2c ar newcap
c2a [ -v ] capability...
```

Description

In some instances it is necessary to deal with capabilities as human readable strings. To do this a special format has been devised known as *ar*-format. It is described in *ar*(L).

A2c coverts *ar*-format to a capability and stores it as *newcap* using the directory primitives.

C2a prints each capability in ar-format on a separate line. The **−v** flag tells *c2a* to prefix each line with the name of the capability, and a tab.

Diagnostics

The error messages of these programs should be self-explanatory. The exit status is 2 in case of an illegal command line. *A2c* will exit with 1 if it cannot append the new capability. *C2a* will exit with 1 if it cannot lookup the capability.

Warning

Since the ar-format contains parentheses, which are special to the shell, the first argument to *a2c* must be quoted.

Examples

To save the capability *foo* in readable form on a file *foo.cap*, type:

```
c2a foo > foo.cap
```

To restore the very same capability, issue:

```
a2c "‘cat foo.cap‘" foo
```

See Also

ar(L).

Name

cal − print a calendar

Synopsis

```
cal [month] year
```

Description

Cal prints a calendar for a month or year. The year can be between 1 and 9999. Note that the year 91 is not a synonym for 1991, but is itself a valid year about 19 centuries ago. The calendar produced is the one used by England and her colonies. Try Sept. 1752, Feb 1900, and Feb 2000. If you do not understand what is going on, look up Calendar, Gregorian in a good encyclopedia.

Example

```
cal 3 1992
```

Prints the following calendar for March 1992.

```
      Mar 1992

 S  M Tu  W Th  F  S
 1  2  3  4  5  6  7
 8  9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
29 30 31
```

Name

cat − concatenate files and write them to *stdout*

Synopsis

```
cat [-u] file ...
```

Description

Cat concatenates its input files and copies the result to *stdout*. If no input file is named, or ‘‘−’’ is encountered as a file name, standard input is used. Output is buffered in 512 byte blocks unless the **−u** flag is given.

Options

−u Use unbuffered output.

Examples

```
cat file
```

Display file on the terminal.

```
cat file1 file2 | lpr
```

Concatenate 2 files and print result.

Name

cc − ACK ANSI C Compiler

Synopsis

```
cc [options] [file ...]
```

Description

Cc is a driver that calls the ANSI C front end and Amsterdam Compiler Kit (see *ack*(U)) back ends to translate a C program into ACK format object code or an Amoeba binary. By default, *cc* generates code for the architecture of the compiler itself. The program *cc* is actually a link to the generic driver of ACK *ack*(U). The only difference is that it is provides a C run-time system (consisting of *libajax.a* and *libamoeba.a*) when it is used to link object files (see *led*(U)). Here, we will only describe the options applicable when compiling C programs.

Options

−A[*file*]
: if *file* is not given, generate a list of *amake*(U) dependencies — consisting of a list of all included files — and write them to the standard output. If *file* is given, the dependencies will be generated on file *file*. Note that the dependency generation takes place in addition to compilation itself.

−D*name=text*
: define *name* as a macro with *text* as its replacement text.

−D*name*
: the same as **−D***name*=1.

−I*dirname*
: insert *dirname* in the list of include directories.

−M[*file*]
: generate dependencies, like with the −A option, only this time generate them in a format suitable as *make*(U) dependencies, i.e., of the form

    src.o: header.h

  where *src.o* is derived from the source file name.

−O call the target optimizer after generating assembly code.

−O*num*
: invoke the EM global optimizer before generating assembly code. The integer argument *num* determines the kind and amount of optimizations being performed. It can range from 1 to 4; the higher the number the more optimization phases.

−U*name*
: get rid of the compiler-predefined macro *name*.

−a suppress warnings and stricts.

−c generate an object file rather than a runnable program. The default name for an object file to be generated is *source.o*. The default name for a runnable program is

a.out.

−c.*suffix*
like **−c**, but generate stop when a file with extension *suffix* has been produced. So **−c.s** causes *cc* to produce an assembly file. (See *ack*(U) for all file types known to ACK.)

−i when generating dependencies, do not include files from */usr/include*.

−o *name*
let the output be called *name* instead of the default *source.o*, or *a.out*.

−p generate code at each procedure entry to call the routine *procentry()*, and at each return to call the routine *procexit()*. These routines are supplied with one parameter, a pointer to a string containing the name of the procedure.

−s suppress stricts.

−w suppress all warning messages. By default, only warning messages complaining about old style — but legal ANSI C — constructions are suppressed.

−wn do not suppress warnings and stricts about old-style C, such as non-prototyped function definitions.

Refer to the ACK ANSI C front end documentation for a specification of the implementation-defined behavior of the ACK ANSI C compiler.

Diagnostics

All warning and error messages are written on standard error output.

Files

*/tmp/** − temporary files

Warnings

Although the C front end running on Amoeba is derived from the ANSI conforming ACK C compiler, the Amoeba headers and libraries are not yet standard conforming, because they are partly adaptations from a pre-ANSI library. Some ANSI library functions are missing, while others will have slightly different semantics. Another problem we are aware of is the current name space pollution. All these things will be fixed in the near future.

See Also

ack(U), amake(U), led(U), make(U).

Name

cdiff − context diff

Synopsis

```
cdiff [-c] old new
```

Description

Cdiff produces a context diff by first running *diff* to find the differences between the files *old* and *new* and then adding context. Some update programs, like patch, can use context diffs to update files, even in the presence of other, independent changes.

Options

−c *n* provide *n* lines of context.

Examples

```
cdiff old new > f
```

Write context diff on *f*.

```
cdiff -c1 old new > f
```

Use only 1 line of context.

Name

cgrep − grep and display context

Synopsis

```
cgrep [-a n] [-b n] [-l n] [-w n] pattern [file] ...
```

Description

Cgrep is a program like grep, except that it also can print a few lines above and/or below the matching lines. It also prints the line numbers of the output.

Options

−a The number of lines to display after the matching line.

−b The number of lines to display before the matching line.

−f Suppress file name in the output.

−l Lines are truncated to this length before comparison.

−n Suppress line numbers in the output.

−w Sets window size. (This is equivalent to **−a***n* **−b***n*).

Example

```
cgrep -w 3 hello file1
```

Print 3 lines of context each way.

Name

chm − change the rights masks in a directory entry

Synopsis

```
chm pathname mask...
```

Description

Chm uses *sp_chmod*(L) to change the rights masks in the directory entry specified by *pathname*. Each rights mask should be specified by one or two hexadecimal digits. The number of mask arguments must be the same as the number of rights-mask columns in the directory entry. (Use: `dir -lc pathname` to see how many rights-mask columns there are in each directory entry.)

If the path name cannot be found in the directory server or insufficient rights are available for the change-mode operation a suitable error message will be printed.

Example

Assuming there are three rights columns, the first for ‘‘owner’’, the second for ‘‘group’’ and the third for ‘‘world’’,

```
chm /home/temp ff 2 0
```

will set the masks so that the owner rights are not restricted (beyond the restrictions imposed by the rights in the capability), the group rights are all turned off except for the second bit (which can be on if it is present in the capability) and the world has no rights, regardless of the rights in the capability.

See Also

del(U), dir(U), get(U), mkd(U), put(U).

Name

chpw − change password

Synopsis

```
chpw [allo-file]
```

Description

Chpw sets the password that you must type to *allo*(A) in order to log in to Amoeba. An encrypted version of your password is kept in the allo file, typically */Environ*. Chpw asks you to type your new password, with input echoing turned off. It then asks you to type it again. If you type a different password, it reports a 'Mismatch' and prompts for a retry. It exits on end-of-file. Otherwise it changes the encrypted password in the file to an encrypted version of the new password.

The optional argument *allo-file* is the name of the file that chpw must change. The default is */Environ*, i.e., your own allo file.

Diagnostics

Some frequent error messages:

`Mismatch`. You retyped a different password.

`Operation not permitted`. The output file cannot be written.

Warnings

Because the encrypted password is stored in a file writable by the user, others with access to your keyboard or with write access to your root directory may change your password without knowing the old password. This will be fixed in a future version of Amoeba.

Examples

To change your own password:

```
chpw
```

To change Mike's password (you need to be a privileged user for this):

```
chpw /super/users/mike/Environ
```

File Format

Chpw does not care much about the format of the input file. It looks for a line beginning with the word `passwd`, surrounded by white space, and replaces the rest of the line with the encrypted password, inserting a tab if necessary.

See Also

allo(A).

Name

clear − clear the screen

Synopsis

```
clear
```

Description

All text is removed from the screen, resulting in an empty screen with the cursor positioned in the upper left-hand corner.

Example

```
clear
```

Clears the screen.

Name

cmp − compare two files

Synopsis

```
cmp [-ls] file1 file2
```

Description

Two files are compared. If they are identical, exit status 0 is returned. If they differ, exit status 1 is returned. If either of the files cannot be opened, exit status 2 is returned. If *file1* is ‘‘−’’, *stdin* is compared to *file2*.

Options

−l Loud mode. Print bytes that differ (in octal)

−s Silent mode. Print nothing, just return exit status

Examples

```
cmp file1 file2
```

Tell whether the files are the same.

```
cmp -l file1 file2
```

Print all corresponding bytes that differ.

See Also

cdiff(U), diff(U).

Name

comm − print lines common to two sorted files

Synopsis

```
comm [-123] file1 file2
```

Description

Two sorted files are read and compared. A three column listing is produced. Lines only in *file1* are in column 1, lines only in *file2* are in column 2 and lines common to both files are in column 3. The file name ‘‘−’’ means *stdin*.

Options

−1 Suppress column 1 (lines only in file1).

−2 Suppress column 2 (lines only in file2).

−3 Suppress column 3 (lines in both files).

Examples

```
comm file1 file2
```

Print all three columns.

```
comm -12 file1 file2
```

Print only lines common to both files.

Name

compress − compress a file using modified Lempel-Ziv coding

Synopsis

```
compress   [-cdfv] [file] ...
uncompress [file.Z] ...
zcat       [file.Z] ...
```

Description

The listed files (or *stdin*, if none are given) are compressed using the Lempel-Ziv algorithm. If the output is smaller than the input, the output is put on *file.Z* or *stdout* if no files are listed. If *compress* is linked to *uncompress*, the latter is the same as giving the **−d** flag. It will produce the file with the same name but without the *.Z* suffix and delete the original compressed file. Similarly, a link to *zcat* decompresses to *stdout*, but the original *.Z* file is not deleted.

Options

−c Put output on *stdout* instead of on *file.Z*.

−d Decompress instead of compress.

−f Force output even if there is no saving.

−v Verbose mode.

Examples

```
compress < infile > outfile
```

Compress *infile* and writes the result on *outfile*.

```
compress x y z
```

Compress the 3 files to *x.Z*, *y.Z* and *z.Z* respectively.

```
compress -d file.Z
```

Decompress *file.Z* to file.

```
uncompress file.Z
```

Equivalent to the previous example.

```
zcat file.Z
```

Decompresses *file.Z* and writes it to *stdout*.

Name

cp − copy file

Synopsis

```
cp file1 file2
cp file ... directory
```

Description

Cp copies one file to another, or copies one or more files to a directory. A file cannot be copied to itself.

Examples

```
cp oldfile newfile
```

Copy *oldfile* to *newfile*.

```
cp file1 file2 /usr/ast
```

Copies two files to a directory.

Name

cpdir − copy a directory and its subdirectories

Synopsis

```
cpdir [-mv] srcdir destdir
```

Description

Cpdir creates the target directory, goes into it, and copies all the files in the source directory to it. When it is done, the target directory contains copies of files in the source directory. Capabilities for all other objects are also copied, and subdirectories are copied recursively.

Options

−v Verbose; cpdir tells what it is doing

−m Merge; just merge in the files, the target directories should already exist.

Example

```
cpdir dir1 dir2
```

Create *dir2* and copy *dir1*'s files into it.

Name

cpp − C preprocessor

Synopsis

```
cpp [options] [inputfile]
```

Description

Cpp reads a file, expands macros and include files, and writes an input file for the C compiler. All output is to standard output. When no input file is provided as argument, *cpp* reads from standard input.

Options

−I*directory*
: add this directory to the list of directories searched for

```
#include "..."
```

and

```
#include <...> commands.
```

Note that there is no space between the **−I** and the directory string. More than one **−I** option is permitted.

−I : end the list of directories to be searched, and also do not look in default places.

−D*name*=*text*
: define *name* as a macro with *text* as its replacement text.

−D*name* : the same as **−D***name*=1.

−U*name*
: undefine the macro name *name*.

−C : leave comments in. By default, C-comments are deleted.

−P : do not generate line directives

−M*n* : set maximum identifier length to *n*.

−d*file* : generate a list of *amake*(U) dependencies – consisting of a list of all included files – on the file specified.

−i : when generating dependencies, do not include files from */usr/include*.

−m : when generating dependencies, generate them in the format suitable as *make*(U) dependencies, i.e.,

```
file.o: file1.h
```

where *file.o* is derived from the source file name.

−undef
: this flag is silently ignored, for compatibility with other preprocessors.

The following names are always available unless undefined:

__FILE__ : The input (or #include) file being compiled (as a quoted string).

__LINE__ The line number being compiled.

__DATE__ The date and time of compilation as a *ctime()* quoted string (the trailing newline is removed).

Warnings

The *cpp* as supplied is not an ANSI C preprocessor. As people have been known to (ab)use the C preprocessor for all kinds of weird things, this is perhaps just for the best. The ACK ANSI Compiler has its own preprocessor built-in.

See Also

ack(U), amake(U), cc(U).

Name

ctags − generates ‘‘tags’’ and (optionally) ‘‘refs’’ files

Synopsis

```
ctags [-r] files...
```

Description

Ctags generates the *tags* and *refs* files from a group of C source files. The *tags* file is used by *Elvis’* ‘‘:tag’’ command, CTRL−] command, and **−t** option. The ‘‘refs’’ file is used by the *ref*(U) program.

Each C source file is scanned for `#define` statements and global function definitions. The name of the macro or function becomes the name of a tag. For each tag, a line is added to the *tags* file which contains:

- the name of the tag
- a tab character
- the name of the file containing the tag
- a tab character
- a way to find the particular line within the file.

The *refs* file is used by the *ref*(U) program, which can be invoked via *elvis’* **K** command. When *ctags* finds a global function definition, it copies the function header into the *refs* file. The first line is flushed against the right margin, but the argument definitions are indented. The *ref* program can search the *refs* file *much* faster than it could search all of the C source files.

The file names list will typically be the names of all C source files in the current directory, like this:

```
ctags -r *.[ch]
```

Options

−r This causes *ctags* to generate both *tags* and *refs*. Without **−r**, it would only generate *tags*.

Warnings

This version of ctags does not parse ANSI-C source code very well. It has trouble recognizing the new-style function definitions.

Files

tags The ‘‘tags’’ file.

refs The ‘‘refs’’ file.

See Also

elvis(U), refs(U).

Name

date − show date and time

Synopsis

```
date [−u] [−c] [−n] [−d dsttype] [−t minutes-west]
      [−a [+|-]sss.fff] [+format] [[yyyy]mmddhhmm[yy][.ss]]
```

Description

(This utility is part of the *localtime* package distributed though USENET. Its main reason for inclusion in Amoeba is the **+format** option.)

Date without arguments writes the date and time to the standard output in the form

```
Thu Feb  8 13:51:09 MET 1990
```

with **MET** replaced by the local time zone's abbreviation (or by the abbreviation for the time zone specified in the **TZ** environment variable if set).

If a command-line argument starts with a plus sign (' + '), the rest of the argument is used as a *format* that controls what appears in the output. In the format, when a percent sign (' % ') appears, it and the character after it are not output, but rather identify part of the date or time to be output in a particular way (or identify a special character to output):

	Sample output	Explanation
%a	Wed	Abbreviated weekday name
%A	Wednesday	Full weekday name
%b	Mar	Abbreviated month name
%B	March	Full month name
%c	03/08/89 14:54:40	Month/day/year Hour:minute:second
%C	Wed Mar 8 14:54:40 1989	a la *asctime*(3)
%d	08	Day of month (always two digits)
%D	03/08/89	Month/day/year (eight characters)
%e	8	Day of month (leading zero blanked)
%h	Mar	Abbreviated month name
%H	14	24-hour-clock hour (two digits)
%I	02	12-hour-clock hour (two digits)
%j	067	Julian day number (three digits)
%k	2	12-hour-clock hour (leading zero blanked)
%l	14	24-hour-clock hour (leading zero blanked)
%m	03	Month number (two digits)
%M	54	Minute (two digits)
%n	\n	newline character
%p	PM	AM/PM designation
%r	02:54:40 PM	Hour:minute:second AM/PM designation
%R	14:54	Hour:minute
%S	40	Second (two digits)
%t	\t	tab character
%T	14:54:40	Hour:minute:second
%U	10	Sunday-based week number (two digits)

%w	3	Day number (one digit, Sunday is 0)
%W	10	Monday-based week number (two digits)
%x	03/08/89	Month/day/year (eight characters)
%X	14:54:40	Hour:minute:second
%y	89	Last two digits of year
%Y	1989	Year in full
%Z	EST	Time zone abbreviation

If a character other than one of those shown above appears after a percent sign in the format, that following character is output. All other characters in the format are copied unchanged to the output; a newline character is always added at the end of the output.

In Sunday-based week numbering, the first Sunday of the year begins week 1; days preceding it are part of ‘‘week 0.’’ In Monday-based week numbering, the first Monday of the year begins week 1.

To set the date, use a command line argument with one of the following forms:

1454	24-hour-clock hours (first two digits) and minutes
081454	Month day (first two digits), hours, and minutes
03081454	Month (two digits, January is 01), month day, hours, minutes
8903081454	Year, month, month day, hours, minutes
0308145489	Month, month day, hours, minutes, year (on System V-compatible systems)
030814541989	Month, month day, hours, minutes, four-digit year
198903081454	Four-digit year, month, month day, hours, minutes

If the century, year, month, or month day is not given, the current value is used. Any of the above forms may be followed by a period and two digits that give the seconds part of the new time; if no seconds are given, zero is assumed.

Options

−u or **−c**
Use GMT when setting and showing the date and time.

−n
Do not notify other networked systems of the time change.

−d dsttype
Set the kernel-stored Daylight Saving Time type to the given value. (The kernel-stored DST type is used mostly by ‘‘old’’ binaries.)

−t minutes-west
Set the kernel-stored ‘‘minutes west of GMT’’ value to the one given on the command line. (The kernel-stored DST type is used mostly by ‘‘old’’ binaries.)

−a adjustment
Change the time forward (or backward) by the number of seconds (and fractions thereof) specified in the *adjustment* argument. Either the seconds part or the fractions part of the argument (but not both) may be omitted. On BSD-based systems, the adjustment is made by changing the rate at which time advances; on System-V-based systems, the adjustment is made by changing the time.

Bugs

Setting the date does not work. Use *adate* (U) instead.

See Also

adate(U), ctime(L), klok(U), posix(L), stdc(L), zdump(U), zic(U).

Name

dd − disk dumper

Synopsis

```
dd [option=value] ...
```

Description

This command is intended for copying partial files. The block size, skip count, and number of blocks to copy can be specified. The options can be used to specify other sizes than the default ones. Where sizes are expected, they are in bytes. However, the letters **w**, **b** or **k** may be appended to the number to indicate words (2 bytes), blocks (512 bytes), or K (1024 bytes), respectively. When *dd* is finished, it reports the number of full and partial blocks read and written.

Options

if=*file*	Input file (default is *stdin)*.
of=*file*	Output file (default is standard output).
ibs=*n*	Input block size (default 512 bytes).
obs=*n*	Output block size (default is 512 bytes).
bs=*n*	Block size; sets ibs and obs (default is 512 bytes).
skip=*n*	Skip *n* input blocks before reading.
seek=*n*	Skip *n* output blocks before writing.
count=*n*	Copy only n input blocks.
conv=lcase	Convert upper case letters to lower case.
conv=ucase	Convert lower case letters to upper case.
conv=swab	Swap every pair of bytes.
conv=noerror	Ignore errors and just keep going.

Examples

```
dd if=x of=y bs=1w skip=4
```

Copy x to y, skipping 4 words.

```
dd if=x of=y count=3
```

Copy three 512-byte blocks.

Name

del − delete a directory entry specified by name

Synopsis

```
del [−df] pathname...
```

Description

For each specified *pathname*, *del* uses *sp_delete* (L) to delete the specified directory entry. Normally, it does not destroy any of the objects referred to by the capabilities in the capability set of the directory entry. Thus any other directory entries referring to the same object(s) are still valid. If there are no other directory entries for one of the objects, it will eventually be destroyed by the garbage-collection process. (This is the usual way for objects to be destroyed, rather than by an explicit user command, since users do not always know whether there are multiple entries for an object.)

If any path name cannot be found in the directory server a suitable error message will be printed.

Options

−d To avoid accidents, this flag must be specified when the directory entry refers to a directory, or when the type of object cannot be determined because its server is not responding.

−f This flag will cause an attempt to destroy each object referred to by one of the capabilities in the set. Whether or not the attempt is fully successful, the directory entry itself will be deleted. It is of course required that the capability includes the right to destroy or modify the object. Also, the directory server rejects attempts to destroy a non-empty directory, regardless of the capability.

Example

```
get /home/oldname | put /home/test/newname
del -d /home/oldname
```

will rename directory */home/oldname* to */home/test/newname*.

See Also

chm(U), dir(U), get(U), mkd(U), put(U).

Name

diff − print differences between two files

Synopsis

```
diff file1 file2
```

Description

Diff compares two files and generates a list of lines telling how the two files differ. Lines may not be longer than 128 characters.

Example

```
diff file1 file2
```

Print differences between *file1* and *file2*.

See Also

cdiff(U), cmp(U).

Name

dir − list information about directory entries

Synopsis

```
dir [-cdklrtux] [pathname ...]
```

Description

For each *pathname*, *dir* displays on standard output information about the specified directory entry, such as rights masks and creation time, and (optionally) about the object referred to by the entry. If no *pathname* is specified it defaults to the current working directory. If *pathname* refers to a directory, the entries in that directory are listed in alphabetic name order, rather than listing the entry that refers to the directory (but see **−d**, below).

If the path name cannot be found in the directory server or insufficient rights are available, a suitable error message will be printed.

Options

−c list the rights masks of each directory entry. The listing begins with a header line in this case. Above the file name(s) are the rights for *delete* and *modify* rights. If either is present it will appear between braces. Then for each column of the directory is printed a heading over the column showing the rights masks per directory entry. The heading printed consists of the name of each column (typically *owner group other*). If you have read permission for the column then an ‘‘*’’ will appear immediately after the column name.

−d if the object is a directory, list information about the entry for the directory object, rather than listing each entry in the directory.

−k print in a human-readable form an approximation of the capability in the directory entry. This form is useful only for certain debugging purposes and cannot be translated back into a valid capability.

−l use *std_info*(L) to obtain an informational string describing the object from the server that owns it. If the object is a directory, this string will show the rights on the directory (after the rights-masking operation is applied).

−r when listing a directory (in the absence of **−d**), sort the directory entries in reverse alphabetic name order, or the reverse of whatever order is specified by the other flags.

−t when listing a directory (in the absence of **−d**), sort the directory entries by their creation date, from newest to oldest.

−u when listing a directory (in the absence of **−d**), use the actual order that the entries appear in the directory, rather than alphabetic order.

−x print three numbers, separated by commas, describing the capability set in the directory entry: the initial member of the suite, the number of current members and the final member of the suite.

Certain *std_info* strings have been encoded to increase the amount of information that can be contained in them. For example, the *std_info* string for a Bullet file begins with the character

‘‘−’’ followed by the size of the file in bytes. The *std_info* string for a Soap directory begins with a ‘‘/’’ followed by the rights that you have in that directory. The ‘‘d’’ stands for *delete* permission, the ‘‘w’’ stands for *write* permission (i.e., modify) and the digits refer to the columns of the directory in which you have read permission. If any of the positions in the string has a ‘‘−’’ character then that right is not present. For kernel directories the information is the same but the initial character identifying the object type is a ‘‘%’’ instead of a ‘‘/’’. For details of other encoded strings see *std_info*(U).

Warnings

Note that if the **−d** flag has *not* been specified and *pathname* is refers to an object for which no server is available then an attempt will be made to locate the server which will result in a time-out of several seconds.

Example

```
dir -lc /home/foo lib/bar
```

might produce the following output:

```
{delete|modify}  owner*    group*    other*
/home/foo     ffffffff  ffffffff  ffffffff  Mar  9 00:31  -     362 bytes
                 owner     group*    other*
lib/bar             ff        ff        ff  Mar 27 17:36  -   73952 bytes
```

The {delete|modify} above */home/foo* says that you have *delete* and *modify* permission for */home*. This is not present for the listing of *lib/bar* so this permission is not present for the directory *lib*. An ‘‘*’’ adjacent to a column name in the header lines implies that you have read permission for that column. The OR of the rights masks in the columns for which you have read permission is used to restrict the capability’s rights before returning it to the user.

See Also

chm(U), del(U), get(U), mkd(U), put(U).

Name

echo − print the arguments

Synopsis

```
echo [-n] argument ...
```

Description

Echo writes its arguments to standard output. They are separated by blanks and terminated with a line feed, unless the **−n** option is present. This command is used mostly in shell scripts.

Examples

```
echo Start Phase 1
```

‘‘Start Phase 1’’ is printed.

```
echo -n Hello
```

‘‘Hello’’ is printed without a line feed.

Name

ed − editor

Synopsis

```
ed file
```

Description

Ed is functionally equivalent to the standard V7 editor, *ed*. It supports the following commands:

(.)a: append
(.,.)c: change
(.,.)d: delete
e: edit new file
f: print name of edited file
(1,$)g: global command
(.) i: insert
(.,.+1)j: join lines together
(.)k: mark
(.)l: print with special characters in octal
(.,.)m: move
(.,.)p: print
q: quit editor
(.) r: read in new file
(.,.)s: substitute
(1,$)v: like g, except select lines that do not match
(1,$)w: write out edited file

Many of the commands can take one or two addresses, as indicated above. The defaults are shown in parentheses. Thus ‘‘a’’ appends to the current line, and ‘‘g’’ works on the whole file as default. The dot refers to the current line. Below is a sample editing session.

Examples

```
ed prog.c
```

Edit *prog.c*.

```
3,20p
```

Print lines 3 through 20.

```
/whole/
```

Find next occurrence of whole.

```
s/whole/while/
```

Replace ‘‘whole’’ by ‘‘while.’’

```
g/Buf/s//BUF/g
```

Replace ‘‘Buf’’ by ‘‘BUF’’ everywhere.

```
w
```

Write the file back.

```
q
```

Exit the editor.

Name

elvis − full screen editor

Synopsis

```
elvis [flags] [+cmd] [files...]
vi    [flags] [+cmd] [files...]
ex    [flags] [+cmd] [files...]
view  [flags] [+cmd] [files...]
```

Description

Elvis is a text editor which emulates *vi* and *ex*.

On Amoeba, you may also install *elvis* under the names *ex*, *vi* and *view*. These extra names would normally be links to elvis; see the *ln*(U) shell command.

When *elvis* is invoked as *vi*, it behaves exactly as though it was invoked as *elvis*. However, if you invoke *elvis* as *view*, then the read-only option is set as though you had given it the **−R** flag. If you invoke *elvis* as *ex*, then *elvis* will start up in the colon command mode instead of the visual command mode, as though you had given it the **−e** flag. If you invoke *elvis* as *edit*, then *elvis* will start up in input mode, as though the **−i** flag was given.

Options

−r To the real *vi*, this flag means that a previous edit should be recovered. *Elvis*, though, has a separate program, called *virec*(U), for recovering files. When you invoke *elvis* with **−r**, *elvis* will tell you to run *virec*.

−R This sets the ‘‘read-only’’ option, so you will not accidentally overwrite a file.

−t *tag* This causes *elvis* to start editing at the given tag.

−m [*file*] *Elvis* will search through *file* for something that looks like an error message from a compiler. It will then begin editing the source file that caused the error, with the cursor sitting on the line where the error was detected. If you do not explicitly name a *file* then *errlist* is assumed.

−e *Elvis* will start up in colon command mode.

−v *Elvis* will start up in visual command mode.

−i *Elvis* will start up in input mode.

+*command* If you use the +*command* parameter, then after the first file is loaded *command* is executed as an *ex* command. A typical example would be

```
elvis +237 foo
```

which would cause *elvis* to start editing *foo* and then move directly to line 237.

Files

/tmp/elv* During editing, *elvis* stores text in a temporary file. For Amoeba, this file will usually be stored in the */tmp* directory, and the first three characters will be

‘‘elv’’. For other systems, the temporary files may be stored someplace else; see the version-specific section of the documentation.

tags — This is the database used by the *:tags* command and the **−t** option. It is usually created by the *ctags* (U) program.

Warnings

There is no LISP support. Certain other features are missing, too.

Auto-indent mode is not quite compatible with the real *vi*. Among other things, 0ˆD and ˆˆD do not do what you might expect.

Long lines are displayed differently. The real *vi* wraps long lines onto multiple rows of the screen, but *elvis* scrolls sideways.

See Also

ctags(U), ref(U), virec(U), elvis(U).

Name

envcap − get a capability from, or list, the capability environment

Synopsis

```
envcap [name]
```

Description

With no arguments, *envcap* lists the names of all the capabilities in the capability environment.

Given the name of one of the capabilities in the environment, *envcap* uses *getcap*(L) to obtain the capability specified by *name* and writes it to standard output, in the form of a singleton capability set suitable as input to the *put* command. No check for the validity of the capabilities in the environment is performed.

Example

```
envcap TTY | put /dev/tty34
```

will install the `TTY` capability of the current process under the name */dev/tty34* in the directory server. If any process then writes to */dev/tty34*, the output will go to the tty that was in use when the envcap command was invoked.

See Also

get(U), getcap(L), put(U).

Name

expand − convert tabs to spaces

Synopsis

```
expand [-t1,t2, ...] [file]
```

Description

Expand replaces tabs in the named files with the equivalent numbers of spaces. If no files are listed, *stdin* is used. The default is a tab every 8 spaces.

Options

−t*n* Tab stop positions at the *n* th character position. If only one tab position is given, the rest are multiples of it.

Example

```
expand -16,32,48,64
```

Expand *stdin* with tabs every 16 columns.

Name

expr − evaluate expression

Synopsis

```
expr expr operator expr ...
```

Description

Expr computes the value of its arguments and writes the result on standard output. The valid operators, in order of increasing precedence, are listed below. Grouped operators have the same precedence.

|
: Or: return the left hand side if it is neither 0 nor ’’, otherwise return the right hand side.

&
: And: return the left hand side if neither left or right hand side are 0 or ’’. Otherwise it returns 0.

<, <=, ==, !=, >=, >
: If both expressions are integer, return the result of the numeric comparison, otherwise do a lexical comparison.

+, −
: Add or subtract two integer values.

*
: Multiply two integer values.

:
: Match the first argument to the second, the latter being interpreted as regular expression. If the regular expression contains a ‘‘\(regexp\)’’ pair, the matched part is returned, otherwise it evaluates to the number of characters matched.

Example

```
x=‘expr + 1‘
```

Add 1 to shell variable x.

```
x=‘expr $path : ’.*/\(.*\)’‘
```

If *$path* is a path name containing one or more ‘‘/’’ characters, *x* will be set to the last component.

Name

factor − factor an integer less than 2**31

Synopsis

```
factor number
```

Description

Factor prints the prime factors of its argument in increasing order. Each factor is printed as many times as it appears in the number.

Example

```
factor 450180
```

Print the prime factors of 450180, i.e.,

```
2 2 3 3 5 41 61
```

Name

false − exit with the value false

Synopsis

```
false
```

Description

This command returns the value false. It is used for shell programming.

See Also

true(U).

Name

fgrep − fast grep

Synopsis

```
fgrep [-c] [-h] [-l] [-n] [-s] [-v] [[-f file] | string] [file] ...
```

Description

Fgrep is essentially the same as *grep*, except that it only searches for lines containing literal strings (no wildcard characters), and it is much faster.

Options

−c Count matching lines and only print count, not the lines.

−f Take strings from file named in following argument.

−h Omit file headers from printout.

−l List file names once only.

−n Each line is preceded by its line number.

−s Status only, no output.

−v Print only lines not matching.

Examples

```
fgrep % prog.c
```

Print lines containing a % sign.

```
fgrep -f pattern prog.c
```

Print all the lines in *prog.c* that contain *pattern*.

Name

file − make a guess as to a file's type based on contents

Synopsis

```
file name ...
```

Description

File reads the first block of a file and tries to make an intelligent guess about what kind of file it is. It understands about archives, C source programs, loadable objects, Amoeba binaries, shell scripts, and English text.

Example

```
file a.out ar.h src
```

This might print:

```
a.out: mc68000 executable
ar.h: C program
src: directory
```

Name

find − find files meeting a given condition

Synopsis

```
find directory expression
```

Description

Find descends the file tree starting at the given directory checking each file in that directory and its subdirectories against a predicate. If the predicate is true, an action is taken. The predicates may be connected by **−a** (Boolean and), **−o** (Boolean or) and **!** (Boolean negation). Each predicate is true under the conditions specified below. In the following, the integer *n* may also be **+n** to mean any value greater than *n*, **−n** to mean any value less than *n*, or just **n** for exactly *n*.

−name *s* true if current file name is *s* (include shell wild cards).

−size *n* true if file size is *n* blocks.

−mtime *n* true if modification time relative to today (in days) is *n*.

−newer *f* true if the file is newer than *f*.

−perm *n* true if the file's permission bits = *n* (*n* is in octal).

−type *x* where *x* is ''d'' for directory of ''f'' for a regular file.

Following the expression can be one of the following, telling what to do when a file is found:

−exec *cmd arg ...* ;
execute a command. {} stands for the current file name.

−print print the file name on standard output.

−ok prompts before executing the command.

Warnings

To prevent the shell from interpreting the semicolon, which is needed to terminate a **-exec** option, it should be quoted (e.g., by prefixing it with a ''\'').

Examples

```
find / -name a.out -print
```

Print all a.out paths.

```
find /home ! -newer f -ok rm {}
```

Ask before removing.

```
find /usr -size +20 -exec mv {} /big \;
```

move files larger than 20 blocks to the directory */big*.

```
find / -name a.out -o -name '*.o' -exec rm {} \;
```

Remove all the file in the system with the name *a.out* and all the files which have the suffix *.o*.

Name

flex − fast lexical analyzer generator

Synopsis

```
flex [-dfirstvFILT -c[efmF] -Sskeleton_file] [filename]
```

Description

flex is a rewrite of *lex* intended to right some of that tool's deficiencies: in particular, *flex* generates lexical analyzers much faster, and the analyzers use smaller tables and run faster.

Options

−d This makes the generated scanner run in *debug* mode. Whenever a pattern is recognized the scanner will write to *stderr* a line of the form:

```
--accepting rule #n
```

Rules are numbered sequentially with the first one being 1.

−f This has the same effect as *lex*'s **−f** flag (do not compress the scanner tables); the mnemonic changes from *fast compilation* to (take your pick) *full table* or *fast scanner*. The actual compilation takes *longer*, since *flex* is I/O bound writing out the big table. This option is equivalent to **−cf** (see below).

−i This instructs *flex* to generate a *case-insensitive* scanner. The case of letters given in the *flex* input patterns will be ignored, and the rules will be matched regardless of case. The matched text given in *yytext* will have the preserved case (i.e., it will not be folded).

−r specifies that the scanner uses the **REJECT** action.

−s causes the *default rule* (that unmatched scanner input is echoed to *stdout*) to be suppressed. If the scanner encounters input that does not match any of its rules, it aborts with an error. This option is useful for finding holes in a scanner's rule set.

−t puts the result on standard output instead of the file *lex.yy.c*.

−v has the same meaning as for *lex* (print to *stderr* a summary of statistics of the generated scanner). Many more statistics are printed, though, and the summary spans several lines. Most of the statistics are meaningless to the casual *flex* user.

−F specifies that the *fast* scanner table representation should be used. This representation is about as fast as the full table representation **(−f)**, and for some sets of patterns will be considerably smaller (and for others, larger). In general, if the pattern set contains both ‘‘keywords’’ and a catch-all, ‘‘identifier’’ rule, such as in the set:

```
"case"    return ( TOK_CASE );
"switch"  return ( TOK_SWITCH );
"default" return ( TOK_DEFAULT );
[a-z]+    return ( TOK_ID );
```

then you are better off using the full table representation. If only the identifier

rule is present and you then use a hash table or some such to detect the keywords, you are better off using **−F**.

This option is equivalent to **−cF** (see below).

−I instructs *flex* to generate an *interactive* scanner. Normally, scanners generated by *flex* always look ahead one character before deciding that a rule has been matched. At the possible cost of some scanning overhead (it is not clear that more overhead is involved), *flex* will generate a scanner which only looks ahead when needed. Such scanners are called *interactive* because if you want to write a scanner for an interactive system such as a command shell, you will probably want the user's input to be terminated with a newline, and without **−I** the user will have to type a character in addition to the newline in order to have the newline recognized. This leads to dreadful interactive performance.

If all this seems to confusing, here is the general rule: if a human will be typing in input to your scanner, use **−I,** otherwise do not; if you do not care about how fast your scanners run and do not want to make any assumptions about the input to your scanner, always use **−I**.

Note, **−I** cannot be used in conjunction with *full* or *fast tables*, i.e., the **−f**, **−F**, **−cf**, or **−cF** flags.

−L instructs *flex* to not generate *#line* directives (see below).

−T makes *flex* run in *trace* mode. It will generate a lot of messages to standard out concerning the form of the input and the resultant non-deterministic and deterministic finite automatons. This option is mostly for use in maintaining *flex*.

−c[efmF]controls the degree of table compression. **−ce** directs *flex* to construct *equivalence classes*, i.e., sets of characters which have identical lexical properties (for example, if the only appearance of digits in the *flex* input is in the character class [0-9] then the digits 0, 1, ..., 9 will all be put in the same equivalence class). **−cf** specifies that the *full* scanner tables should be generated − *flex* should not compress the tables by taking advantages of similar transition functions for different states. **−cF** specifies that the alternate fast scanner representation (described above under the **−F** flag) should be used. **−cm** directs *flex* to construct *meta-equivalence classes*, which are sets of equivalence classes (or characters, if equivalence classes are not being used) that are commonly used together. A lone **−c** specifies that the scanner tables should be compressed but neither equivalence classes nor meta-equivalence classes should be used.

The options **−cf** or **−cF** and **−cm** do not make sense together − there is no opportunity for meta-equivalence classes if the table is not being compressed. Otherwise the options may be freely mixed.

The default setting is **−cem** which specifies that *flex* should generate equivalence classes and meta-equivalence classes. This setting provides the highest degree of table compression. You can trade off faster-executing scanners at the cost of larger tables with the following generally being true:

```
slowest             smallest
          -cem
          -ce
          -cm
          -c
          -c{f,F}e
          -c{f,F}
fastest             largest
```

−**S**skeleton_file overrides the default skeleton file from which *flex* constructs its scanners. You will never need this option unless you are doing *flex* maintenance or development.

Incompatibilities With Lex

flex is fully compatible with *lex* with the following exceptions:

- There is no run-time library to link with. You need not specify **−ll** when linking, and you must supply a main program. (Hacker's note: since the *lex* library contains a *main()* which simply calls *yylex()*, you actually *can* be lazy and not supply your own main program and link with **−ll**.)
- *lex*'s *%r* (Ratfor scanners) and *%t* (translation table) options are not supported.
- The do-nothing **−n** flag is not supported.
- When definitions are expanded, *flex* encloses them in parentheses. With *lex*, the following

  ```
  NAME    [A-Z][A-Z0-9]*
  %%
  foo{NAME}?      printf("Found it\n");
  %%
  ```

  will not match the string ''foo'' because when the macro is expanded the rule is equivalent to ''foo[A-Z][A-Z0-9]*?'' and the precedence is such that the ''?'' is associated with ''[A-Z0-9]*''. With *flex*, the rule will be expanded to ''foo([A-z][A-Z0-9]*)?'' and so the string ''foo'' will match.
- *yymore* is not supported.
- The undocumented *lex*-scanner internal variable *yylineno* is not supported.
- If your input uses *REJECT*, you must run *flex* with the **−r** flag. If you leave out the flag, the scanner will abort at run-time with a message that the scanner was compiled without the flag being specified.
- The *input()* routine is not redefinable, though may be called to read characters following whatever has been matched by a rule. If *input()* encounters and end-of-file the normal *yywrap()* processing is done. A ''real'' end-of-file is returned as *EOF*.

  Input can be controlled by redefining the `YY_INPUT` macro. `YY_INPUT`'s calling sequence is

  ```
  YY_INPUT(buf, result, max_size)
  ```

Its action is to place up to *max_size* characters in the character buffer *buf* and return in the integer variable *result* either the number of characters read or the constant YY_NULL (0 on UNIX systems) to indicate EOF. The default YY_INPUT reads from the file-pointer *yyin* (which is by default *stdin*), so if you just want to change the input file, you need not redefine YY_INPUT − just point *yyin* at the input file.

A sample redefinition of YY_INPUT (in the first section of the input file):

```
%{
#undef YY_INPUT
#define YY_INPUT(buf,result,max_size) ú
    result = (buf[0] = getchar()) == EOF ? YY_NULL : 1;
%}
```

You also can add in things like counting keeping track of the input line number this way; but do not expect your scanner to go very fast.

- *output* is not supported. Output from the ECHO macro is done to the file-pointer *yyout* (default *stdout*).
- Trailing context is restricted to patterns which have either a fixed-sized leading part or a fixed-sized trailing part. For example, ‘‘a*/b’’ and ‘‘a/b*’’ are okay, but not ‘‘a*/b*’’. This restriction is due to a bug in the trailing context algorithm given in *Principles of Compiler Design* (and *Compilers − Principles, Techniques, and Tools*) which can result in mismatches. Try the following *lex* program

```
%%
x+/xy           printf("I found \"%s\"\n", yytext);
```

on the input ‘‘xxy’’. If you must have arbitrary trailing context, you can use *yyless* to effect it.

- *flex* reads only one input file, while *lex*’s input is made up of the concatenation of its input files.

Enhancements

- *Exclusive start-conditions* can be declared by using *%x* instead of *%s*. These start-conditions have the property that when they are active, *no other rules are active*. Thus a set of rules governed by the same exclusive start condition describe a scanner which is independent of any of the other rules in the *flex* input. This feature makes it easy to specify ‘‘mini-scanners’’ which scan portions of the input that are syntactically different from the rest (e.g., comments).
- *flex* dynamically resizes its internal tables, so directives like ‘‘%a 3000’’ are not needed when specifying large scanners.
- The scanning routine generated by *flex* is declared using the macro *YY_DECL*. By redefining this macro you can change the routine’s name and its calling sequence. For example, you could use:

```
#undef YY_DECL
#define YY_DECL float lexscan( a, b ) float a, b;
```

to give it the name *lexscan*, returning a float, and taking two floats as arguments.

- *flex* generates *#line* directives mapping lines in the output to their origin in the input file.
- You can put multiple actions on the same line, separated with semi-colons. With *lex*, the following

```
foo    handle_foo(); return 1;
```

  is truncated to

```
foo    handle_foo();
```

  flex does not truncate the action. Actions that are not enclosed in braces are terminated at the end of the line.
- Actions can be begun with %{ and terminated with %}. In this case, *flex* does not count braces to figure out where the action ends − actions are terminated by the closing *%}*. This feature is useful when the enclosed action has extraneous braces in it (usually in comments or inside inactive #ifdef's) that throw off the brace-count.
- All of the scanner actions (e.g., *ECHO*, *yywrap*...) except the *unput()* and *input()* routines, are written as macros, so they can be redefined if necessary without requiring a separate library to link to.

Diagnostics

flex scanner jammed
: A scanner compiled with **−s** has encountered an input string which was not matched by any of its rules.

flex input buffer overflowed
: A scanner rule matched a string long enough to overflow the scanner's internal input buffer (as large as *BUFSIZ* in ''stdio.h''). You can edit *flexskelcom.h* and increase *YY_BUF_SIZE* and *YY_MAX_LINE* to increase this limit.

REJECT used and scanner was not generated using −r
: Just like it sounds. Thw scanner uses *REJECT*. You must run *flex* on the scanner description using the **−r** flag.

old-style lex command ignored
: The *flex* input contains a *lex* command (e.g., ''%n 1000'') which is being ignored.

Bugs

Use of *unput()* or *input()* trashes the current *yytext* and *yyleng*.

Use of *unput()* to push back more text than was matched can result in the pushed-back text matching a beginning-of-line ('^') rule even though it did not come at the beginning of the line.

Nulls are not allowed in *flex* inputs or in the inputs to scanners generated by *flex*. Their presence generates fatal errors.

Do not mix trailing context with the '|' operator used to specify that multiple rules use the same action. That is, avoid constructs like:

```
foo/bar      |
bletch       |
bugprone     { ... }
```

They can result in subtle mismatches. This is actually not a problem if there is only one rule using trailing context and it is the first in the list (so the above example will actually work okay). The problem is due to fall-through in the action switch statement, causing non-trailing-context rules to execute the trailing-context code of their fellow rules. This should be fixed, as it is a nasty bug and not obvious. The proper fix is for *flex* to spit out a FLEX_TRAILING_CONTEXT_USED #define and then have the backup logic in a separate table which is consulted for each rule-match, rather than as part of the rule action. The place to do the tweaking is in *add_accept()* − any kind soul want to be a hero?

The pattern:

```
x{3}
```

is considered to be variable-length for the purposes of trailing context, even though it has a clear fixed length.

Due to both buffering of input and read-ahead, you cannot intermix calls to, for example, *getchar()* with *flex* rules and expect it to work. Call *input()* instead.

The total table entries listed by the **−v** flag excludes the number of table entries needed to determine what rule has been matched. The number of entries is equal to the number of DFA states if the scanner was not compiled with **−r**, and greater than the number of states if it was.

The scanner run-time speeds have not been optimized as much as they deserve. Van Jacobson’s work shows that the can go quite a bit faster still.

Files

flex.skel skeleton scanner *flex.fastskel* skeleton scanner for **−f** and **−F** *flexskelcom.h* common definitions for skeleton files *flexskeldef.h* definitions for compressed skeleton file *fastskeldef.h* definitions for **−f**, **−F** skeleton file.

See Also

M. E. Lesk and E. Schmidt, *LEX − Lexical Analyzer Generator*.

Name

fold − fold long lines

Synopsis

```
fold [-n] [file] ...
```

Description

Fold takes copies its input from the named file (or *stdin*, if none is specified) to standard output. However, lines longer than the given maximum length (default 80) are broken into multiple lines of the maximum length by inserting new line characters.

Options

−n *n* is a decimal integer specifying how long the output lines should be.

Examples

```
fold -60
```

Fold *stdin* to 60 characters.

```
fold -1 file
```

Fold file to 80 characters.

Name

fortune − print a fortune

Synopsis

```
fortune
```

Description

Fortune picks a fortune at random from the fortunes file, */usr/lib/fortune.dat*, and prints it. This file consists of pieces of text separated by a line containing only %%.

Example

```
fortune
```

This might print:

```
First Law of Socio-Genetics:
Celibacy is not hereditary.
```

or

```
Structured Programming supports the law of the excluded muddle.
```

Name

fromb − read a file from the bullet server

Synopsis

```
fromb [-o offset] [-s size] filename
```

Description

Fromb reads the bullet file named *filename* in one read and writes its contents to standard output. If the file name cannot be found in the directory server or the capability is invalid a suitable error message will be printed. If the offset is greater than the file size it will print an error message and exit. If the offset is valid and *offset* plus *size* is greater than the file size it will read up until the end of the file.

Options

−o offset starts reading the file at *offset* bytes from the start of the file. The default is 0.

−s size says to read a maximum of *size* bytes. The default is the whole file.

Example

```
fromb /home/textfile | wc -l
```

will read the file */home/textfile* and pipe it through the *wc* command and print the number of lines in the file.

See Also

bsize(U), cat(U).

Name

get − get a capability set specified by name

Synopsis

```
get pathname
```

Description

get uses *sp_lookup* (L) to obtain the capability set specified by *pathname* and writes it on standard output. If the path name cannot be found in the directory server a suitable error message will be printed. No check for the validity of the capabilities in the set is performed.

Example

```
get /home/oldname | put /home/newname
```

will install */home/newname* in the directory server as a synonym for */home/oldname*.

See Also

chm(U), dir(U), mkd(U), del(U), put(U).

Name

grep − search a file for lines containing a given pattern

Synopsis

```
grep [-ensv] pattern [file] ...
```

Description

Grep searches one or more files (by default, *stdin*) and selects out all the lines that match the pattern. All the regular expressions accepted by mined are allowed. In addition, ‘‘+’’ can be used instead of ‘‘*’’ to mean 1 or more occurrences, ‘‘?’’ can be used to mean 0 or 1 occurrences, and ‘‘|’’ can be used between two regular expressions to mean either one of them. Parentheses can be used for grouping. If a match is found, exit status 0 is returned. If no match is found, exit status 1 is returned. If an error is detected, exit status 2 is returned.

Options

−e **−e** pattern is the same as pattern

−n Print line numbers

−s Status only, no printed output

−v Select lines that do not match

Examples

```
grep mouse file
```

Find lines in file containing mouse.

```
grep [0-9] file
```

Print lines containing a digit.

Name

head − print the first few lines of a file

Synopsis

```
head [-n] [file] ...
```

Description

The first few lines of one or more files are printed. The default count is 10 lines. The default file is *stdin*.

Options

−n Specifies how many lines to print.

Examples

```
head -6
```

Print first 6 lines of *stdin*.

```
head -1 file1 file2
```

Print first line of *file1* and the first line of *file2* plus a little context information.

Name

ic − integer calculator

Synopsis

```
ic [expression]
```

Description

Ic is a reverse-Polish notation calculator that works on 32-bit integers. It starts out by computing the expression given as an argument, if any, and then expects keyboard input. As an example, to compute ‘‘23+5’’ one first converts this to reverse Polish, ‘‘23 5+’’. After the calculator starts, type ‘‘23’’ followed by a carriage return. Then type ‘‘5’’ and another carriage return. Finally type ‘‘+’’ to see the result, 28 displayed on the stack. Other operations work the same way. The calculator can use other radices for input and output, and has registers that can be stored and loaded. The *h* command gives the help menu.

Examples

```
ic
```

Start the calculator.

```
ic 250 300+
```

Start calculator with 550 on the stack.

Name

kill − send a signal to a process

Synopsis

```
kill [-sig] process
```

Description

A signal is sent to a given process. By default signal 15 (`SIGTERM`) is sent. Process 0 means all the processes in the sender's process group.

Options

−**sig** *sig* is a number specifying the signal number to send.

Examples

```
kill 35
```

Send signal 15 to process 35.

```
kill -9 40
```

Send signal 9 to process 40.

```
kill -2 0
```

Send signal 2 to whole process group.

Name

klok − alarm clock window

Synopsis

```
klok [-s] [-m] [-a hhmm] [-t title] [stdwin-options]
```

Description

This program requires an X11 window server. It creates one window in which it displays an analog clock with two or three hands, and a menu. In the corners of the window some extra information is visible: top left the date, top right an AM/PM indicator, bottom left an alarm indicator if the alarm is set.

When the alarm goes off, the window changes to inverse video and starts beeping. The beeps are repeated every time a hand moves (once a minute, or once a second if the third hand is shown). To turn this mode off, you can click the mouse anywhere in the window, select the 'OK alarm' menu item, or change or clear the alarm time with the 'Set alarm...' or 'Clear alarm' menu item. The alarm also turns itself off after five minutes. Unless the alarm time is changed or cleared, the alarm remains set and will go off again in 24 hours.

The *klok* menu has the following items:

Set alarm...
: Brings up a dialogue box in which you can edit the alarm time.

Clear alarm
: Clears the alarm time; the alarm will not go off.

OK alarm
: Can be used to silence the alarm when it is going off.

Seconds hand
: Toggles the presence of the third (seconds) hand.

Quit
: Closes the window and quits the application.

The menu shortcuts indicated in the menu (Meta-S to set the alarm, etc.) work without the Meta key as well.

By dragging the minutes or seconds hand, the time can be adjusted. This may require special privileges. To cancel a drag started accidentally, press any key while the mouse button is still down. When the hours hand is dragged, minutes and seconds are left unchanged when the time is set.

Options

−s Turns on the third (seconds) hand initially.

−m Turns off the third hand initially; this is also the default.

−a hhmm
: Sets the alarm initially at **hhmm** (hours, minutes, on a 24-hour clock).

−t title
: Sets the window title; the default is the program name ('klok').

More options are described in the X11 stdwin manual (**−display**, **−geometry** etc.); resources also work as described there.

Example

To start the clock in the background:

```
klok &
```

See Also

adate(U), date(U).

Name

leave − warn when it is time to go home

Synopsis

```
leave [ [+] hh[:]mm]
```

Description

Leave sets an alarm clock to a specified time and issues a warning 5 minutes before, 1 minute before, and at the time to leave. It then keeps issuing warnings every minute for 10 minutes, then quits. If no time is provided, the program prompts for one.

Examples

```
leave 1500
```

Issue a warning at 2:55 p.m.

```
leave 10:00
```

Issue a warning at 9:55 a.m.

```
leave + 30
```

Issue a warning in 25 minutes.

Name

listen − listen for messages from user-to-user message service

Synopsis

```
listen
```

Description

Listen is the output half of a simple user interface to Amoeba's user-to-user message service. It listens for messages sent to the user that invoked it, either directly or though ''topics'' in which the user is interested. It prints each incoming message on standard output; messages without a trailing newline have one appended. It should normally run in the background as long as a user is logged in. A user may start multiple listen servers; each server will receive all messages directed to the user or subscribed topics.

Users send messages with *tell*(U). They manage the list of topics in which they are interested with (un)subscribe, see *topics*(L). The actual message transfer is mediated by the multicast server, see *mc*(A) and *mc*(L).

Files

/profile/ms/relay − is the multicast group used to relay all messages to the current user.

Diagnostics

During initialization, complaints are possible when various needed system directories do not exist. When invoked for the first time ever for a particular user, the message ''creating relay group'' is printed.

During execution, one possible message is ''can't join relay group,'' indicating problems with the multicast group used to relay messages to this user. The cause may be that the multicast server is down, or that the relay group has been destroyed. In the former case the listener sticks around, expecting the multicast server to be restarted; in the latter case it exits (it will not create a new relay group).

When a message with a non-zero type is received, a warning is printed on *stdout* before the message is printed.

Warning

The listener does not filter out escape characters, so nasty users may send messages that put your terminal in a weird state, spy on your keyboard, etc. If you can, it is best to run it in a separate window.

Examples

To start the listen server in the background when you log in, put the following in your shell *.profile*:

```
listen &
```

UNIX X users can start the server in a tiny window of its own. For example,

```
xterm -geometry 80x8-0-0 -e listen &
```

This normally done through the shell script *xlisten* (U).
Amoeba users can do the same as follows:

```
aterm -geometry -0-0 -r 8 /bin/listen &
```

See Also

mc(A), mc(L), tell(U), topics(U), xlisten(U).

Name

ln − create a link to a file

Synopsis

```
ln file [name]
```

Description

A directory entry is created for name. The entry points to file. Henceforth, name and file can be used interchangeably. If name is not supplied, the last component of file is used as the link name.

Examples

```
ln file newname
```

Make newname a synonym for file.

```
ln /usr/games/chess
```

Create a link called *chess*.

Name

look − look up words in dictionary

Synopsis

```
look [-f] prefix[/suffix] [dictionary]
```

Description

Look takes a *prefix* and/or *suffix* and searches */usr/lib/dictionary* or the specified dictionary for all words that have that prefix or suffix The words are printed.

Options

−f Fold upper case letters to lower case.

Examples

```
look ard
```

Print words starting with *ard*.

```
look /bing
```

Print words ending with *bing*.

```
look -f f/ar
```

Print words starting with *f*, ending with *ar*.

Name

ls − list the contents of a directory

Synopsis

```
ls [-ACFRadfgilrstuwx] name ...
```

Description

For each file argument, list it. For each directory argument, list its contents, unless **-d** is present. When no argument is present, the working directory is listed.

Options

−A All entries are listed, except . and ..
−C Multicolumn listing (default).
−F Put / after directory names.
−R Recursively list subdirectories.
−a All entries are listed, even file names starting with a ‘.’.
−c Like **−c** except i-node change time is used.
−d Do not list contents of directories.
−f List argument as unsorted directory.
−g Group id given instead of user id.
−l Long listing: mode, links, owner, size and time.
−r Reverse the sort order.
−s Give size in blocks (including indirect blocks).
−t Sort by time, latest first.
−u Use last usage time instead of modification time.

Examples

```
ls -l
```

List files in current directory.

```
ls -Rs /home/bin
```

List */home/bin* recursively with sizes.

See Also

dir(U).

Name

m4 − macro processor

Synopsis

```
m4 [-D name = value] [-U name]
```

Description

M4 is a general-purpose macro processor. It has been used to implement programming languages, such as RATFOR.

Options

−D name = value
Define a symbol.

−U name
Undefine a symbol.

Example

```
m4 < m4test
```

Name

make − a program for maintaining medium-sized programs

Synopsis

```
make [-f file] [ikns] [option] ... [target]
```

Description

Make is a program that is normally used for developing medium-sized programs consisting of multiple files. It keeps track of which object files depend on which source and header files. When called, it does the minimum amount of recompilation to bring the target file up to date.

Options

−f Use *file* as the makefile.

−i Ignore status returned by commands.

−n Report, but do not execute.

−p Print macros and targets.

−q Question up-to-dateness of target.

−r Rule inhibit; do not use default rules.

−s Silent mode.

−t Touch files instead of making them.

The file dependencies are expected in *makefile* or *Makefile*, unless another file is specified with **-f**. Make has some default rules built in, for example, it knows how to make *.o* files from *.c* files. Here is a sample makefile.

```
d=/home/test      # d is a macro
program: head.o tail.o# program depends on these
         cc -o program head.o tail.o# tells how to make program
         echo Program done. # announce completion

head.o: $d/def.h head.c# head.o depends on these
tail.o: $d/var.h tail.c# tail.o depends on these
```

A complete description of make would require too much space here. For more information, see Feldman (1979). Many books on UNIX also discuss *make*.

Warnings

Make uses a file's modification time to see if it is ''out of date.'' As the Soap directory server currently does not provide a very precise file modification time, problems may arise when a file is to be recompiled, shortly after it has been changed. Amoeba's own configuration manager, *amake*(U), does not have this problem, since it records the actual capabilities. before using *make*, look at the manual page for *amake* to see if it is suitable for your application.

Examples

```
make kernel
```

Make *kernel* up to date.

```
make -n f file
```

Tell what needs to be done.

See Also

amake(U).

Name

makecap − Create a random capability

Synopsis

```
makecap [-r rights] [-o objectnumber] [capname]
```

Description

Makecap is used to create random capabilities. The server port and the check word in the capability are chosen at random, and it is extremely unlikely that the capability points to any existing object. The new capability can be stored in the directory server under the name *capname*. If the *capname* argument is not present or is ‘‘−’’, then it writes the capability on standard output.

Options

−o With the **−o** option, one can specify the object number in the capability. By default, this is zero.

−r With the **−r** option, one can specify the rights bits in the capabilities. By default, the capability has all rights bits set.

The default base for *objectnumber* and *rights* is decimal, but prefixing the number with a ‘‘0x’’ will change the base to hexadecimal. If you give invalid number specifications they will be interpreted as 0.

Diagnostics

If either the rights or the object number are out of range, makecap reports a failure in the function *prv_encode*. Other error messages should be self-explanatory.

Example

To create two random capabilities ‘‘foo’’ and ‘‘bar’’, type:

```
makecap foo bar
```

See Also

prv_encode(L), uniqport(L).

Name

makeproto − modifies the output of mkproto to be more useful

Synopsis

```
makeproto [-n] [-s] [-p] [file] ...
```

Description

Makeproto is a shell script which calls *mkproto* to generate ANSI C prototypes for a C source file and then modifies the output of *mkproto* by deleting lines containing the words PRIVATE or LOCAL and converting the word PUBLIC to EXTERN. The result is generated on standard output. If no file name is specified in the command line it reads from standard input. The options are those of *mkproto*.

Example

```
makeproto foo.c > foo.pro
```

will generate ANSI C function prototypes for the routines in *foo.c* and writes them to the file *foo.pro*.

See Also

mkproto(U).

Name

man − display manual page

Synopsis

```
man [man_directory] [digit] [name] ...
```

Description

Man is a program that displays manual pages. When called with a program name, it displays the manual page for that program. When the digit *k* is given as an argument, the file */usr/man/mank* is used instead of the default, */usr/man/man1*. When no name is given (or just a digit), the list of valid entries is displayed. The arrows can be used to select an entry, and <return> can be used to display the selected entry. To exit the program type ‘‘Q’’ or ‘‘q’’. A directory name can be given to override the use of */usr/man*.

Examples

```
man
```

Display main index.

```
man cdiff
```

Display man page for *cdiff*(U).

```
man fork
```

Display man page for *fork*(L).

```
man 3
```

Display the part 3 man pages.

Name

mkd − make a new directory

Synopsis

```
mkd pathname...
```

Description

Mkd uses *sp_mkdir* (L) to create new directories with the given *pathnames*.

Example

```
mkd /home/newdir
```

will create a new directory named *newdir*, under the */home* directory, which must exist before this command is invoked.

See Also

chm(U), dir(U), get(U), del(U), put(U).

Name

mkdir − make a directory

Synopsis

```
mkdir directory ...
```

Description

The specified directory or directories are created.

Examples

```
mkdir dir
```

Create *dir* in the current directory.

```
mkdir /home/dir
```

Create the directory *dir* in the directory */home*.

See Also

mkd(U).

Name

mkproto − make ANSI C function prototypes for a C source file

Synopsis

```
mkproto [-n] [-s] [-p] [file] ...
```

Description

Mkproto takes as input one or more C source code files, and produces as output (on the standard output stream) a list of function prototypes (a la ANSI) for the external functions defined in the given source files. This output, redirected to a file, is suitable for #including in a C source file.

The function definitions in the original source may be either ‘‘old-style’’ (in which case appropriate prototypes are generated for the functions) or ‘‘new-style’’ (in which the definition includes a prototype already).

Options

−n causes the line number where each function was defined to be prepended to the prototype declaration as a comment.

−s causes prototypes to be generated for functions declared ‘‘static’’ as well as extern functions.

−p causes the prototypes emitted to be only readable by ANSI compilers. Normally, the prototypes are ‘‘macro-ized’’ so that compilers with __STDC__ not defined don’t see them.

If files are specified on the command line, then a comment specifying the file of origin is emitted before the prototypes constructed from that file. If no files are given, then no comments are emitted and the C source code is taken from the standard input stream.

Warnings

Mkproto is easily confused by complicated declarations, such as

```
int ((*signal())()) { ...
```

or

```
struct foo { int x, y; } foofunc() { ...
```

Float types are not properly promoted in old style definitions, that is,

```
int test(f) float f; { ...
```

should (because of the default type conversion rules) have prototype

```
int test(double f);
```

rather than the incorrect

```
int test(float f);
```

generated by *mkproto*.

Some programs may need to be run through the preprocessor before being run through *mkproto*. The **−n** option is unlikely to work as desired on the output of a preprocessor.

Typedef'd types are not correctly promoted. For example, for

```
typedef schar char; int foo(x) schar x;...
```

mkproto incorrectly generates the prototype

```
int foo(schar x)
```

rather than the (correct)

```
int foo(int x)
```

Functions named ‘‘inline’’ with no explicit type qualifiers are not recognized.

Note

There is no warranty for this program (as noted above, it is guaranteed to break sometimes anyway).

Example

```
mkproto fred.c > fred.pro
```

will produce the file *fred.pro* which contains the ANSI C prototypes for all the externally visible functions defined in *fred.c*.

See Also

cc(U).

Name

Modula-2 − ACK Modula-2 compiler

Synopsis

```
m2 [options] file ...
```

Description

This document provides a short introduction to the use of the ACK Modula-2 compiler. It also tells you where to find definition modules for ‘‘standard’’ modules.

File Names

Usually, a Modula-2 program consists of several definition and implementation modules, and one program module. Definition modules must reside in files with names having a ‘‘.def’’ suffix. Implementation modules and program modules must reside in files having a ‘‘.mod’’ suffix.

The name of the file in which a definition module is stored must be the same as the module-name, apart from the extension. For historic reasons, the compiler truncates to 10 characters the basename of the definition modules it tries to read. So, given an IMPORT declaration for a module called *LongModulName*, the compiler will try to open a file called *LongModulN.def*. The requirement does not hold for implementation or program modules, but is certainly recommended.

Calling the compiler

The easiest way to call the compiler is to let the *ack*(U) program do it for you. So, to compile a program module *prog.mod*, just call

```
m2 −mmach prog.mod [ objects of implementation modules ]
```

where *mach* is one of the target machines of ACK. If there are no errors, this will produce a binary program *prog* that can be run on machine of kind *mach*. When no **−m** option is given, the architecture of the driver itself is assumed. The program *m2* is actually a link to the generic driver *ack*, which can be used to compile any type of source file supported by ACK (see *ack*(U) for a full list of options available).

To compile an implementation module, use the **−c** flag to produce a *.o* file. Definition modules can not be compiled; the compiler reads them when they are needed.

Definition Modules

‘‘Standard’’ definition modules can be found in the directory */public/module/ack/lib/m2*. When the compiler needs a definition module, it is first searched for in the current directory, then in the directories given to it by the **−I** flag in the order given, and then in the directory mentioned above.

Options

The *ack*(U) program recognizes (among others) the following flags, that are passed to the Modula-2 compiler:

−I*dirname*
append *dirname* to the list of directories where definition modules are looked for.

−I do not look in the directory */public/module/ack/lib/m2*.

−M*n* set maximum identifier length to *n*. The minimum value of *n* is 14, because the keyword ‘‘IMPLEMENTATION’’ is that long.

−d*file* generate a list of *amake*(U) dependencies — consisting of a list of all included files — on the file specified.

−n do not generate EM register messages. The user-declared variables will not be stored into registers on the target machine.

−w*classes*
suppress warning messages whose class is a member of *classes*. Currently, there are three classes: **O**, indicating old-fashioned use, **W**, indicating ‘‘ordinary’’ warnings, and **R**, indicating restricted Modula-2. If no *classes* are given, all warnings are suppressed. By default, warnings in class **O** and **W** are given.

−W*classes*
allow for warning messages whose class is a member of *classes*.

−x make all procedure names global, so that a debugger understands them.

−Xs make INTEGER ranges symmetric, that is, MIN(INTEGER) = –MAX(INTEGER). This is useful for interpreters that use the ‘‘real’’ MIN(INTEGER) to indicate ‘‘undefined.’’

Diagnostics

All warning and error messages are written on standard error output.

Files

/public/module/ack/bin/`arch`*/m2* − the compiler driver.

/public/module/ack/lib/front/`arch`*/em_m2* − the Modula-2 front end.

See Also

ack(U), amake(U).

Name

monitor − monitor a running server

Synopsis

```
monitor name [ subcommand [arg ... ] ]
```

Description

Monitor is used to get the monitor data being generated by a server. Monitoring data consists of a string which identifies an event, followed by a count of the number of times that event has occurred. (See *monitor*(L) for more details.) *Monitor* without a *subcommand* argument gets the capability *name* from the directory service and gets the monitoring data from the server whose port is in the capability. It then prints out all events that occurred in the server together with the counts of the number of times that they have occurred.

It is possible to switch the server's monitor into a mode where it will keep events in a circular buffer in the order in which they occurred. This requires more work by the server. The subcommand *circbuf on* tells the server to start this mode of operation. Thereafter the subcommand *circbuf* without arguments will print a list of events, since the last *circbuf* monitor request. As one would expect *circbuf off* turns this collecting off. Intervening calls to the monitor without the *circbuf* subcommand will return the normal statistics.

The subcommand *reset* will reset all counters to zero in the monitored server.

No meaningful results will be obtained if the server did not do any monitoring.

Diagnostics

Error messages arise because either the server is not running or the server is not doing monitoring. In the latter case you will probably get the error message:

```
monitor: Couldn't get monitor data: invalid capability
```

Example

```
monitor ''
```

will give the monitor data for the directory server, since `''` is your current directory and the server port in your home directory capability is that of the directory server.

```
monitor /dev/session circbuf on
```

will start circular buffer monitoring on the session server. Subsequent

```
monitor /dev/session circbuf
```

commands will print monitor data for the session server events that happened since the previous execution of the command.

```
monitor /dev/session reset
```

will set all the monitor counters to zero in the session server.

See Also

monitor(L).

Name

more − a pager

Synopsis

```
more [-cdflpsu] [-n] [+n] [+/pattern] [file] ...
```

Description

More is a pager that allows one to examine files. This program is patterned after one originally produced at the University of California, Berkeley. When more starts up, it displays a screen full of information from the first file in its list, and then pauses for one of the following commands.

Options

−d Display prompt message at each pause.

−f Do not fold lines.

−l Do not treat CTRL-L as form feed.

−p Page mode. Do not scroll.

−s Suppress multiple blank lines.

−u Use escape sequences for underlining.

Below is a description of interactive commands that can be given to more. In this description, ‘‘#’’ represents an integer telling how many of something.

<space>	Display next page
<return>	Display next line
CTRL-B	Go backward half a screen full
CTRL-D	Go forward half a screen full
CTRL-L	Redisplay the screen
#<space>	Go forward # lines
=	Print current line number
.	Repeat previous command
’	(single quote) Go back to start of last search
!	Escape to a shell
#/<expr>	Go to #-th occurrence of <expr>
:f	Display current file name and line number
#:n	Skip forward # files
#:p	Skip backward # files
b	Go backward half a screen full
d	Go forward half a screen full
#f	Skip # screen fulls
h	Display /usr/lib/more.help

#n	Go to #-th occurrence of last <expr>
q	Quit more
Q	Quit more
#s	Skip # lines
v	Try to execute */usr/bin/vi*
#z	Go forward # lines and set screen size to #

For the benefit of users who always want to use certain flags when calling more, the string environment variable `MORE` can be set to the desired default flags, for example, `MORE`=‘‘−p’’.

Examples

```
more file
```

Display file on the screen.

```
more -p file1 file2
```

Display two files in page mode.

```
more -10 file
```

Use a 10 line window

```
more +/begin file
```

Hunt for the string ‘‘begin’’ in *file*.

See Also

yap(U).

Name

mv − move or rename a file

Synopsis

```
mv file1 file2
mv file ... directory
```

Description

Mv moves one or more files from one place in the file system to another. If the old path and new path are on the same name server, it is done by linking and unlinking, otherwise by copying.

Examples

```
mv oldname newname
```

Move the object *oldname* to *newname*.

```
mv file1 file2 /public/users/ast/home
```

Move the two objects *file1* and *file2* to the directory */public/users/ast/home*.

Name

oc − command for invoking the Orca compiler

Synopsis

```
oc file
```

Description

oc is a shell-script for compiling Orca programs into object files. The script invokes the appropriate passes of the Orca compiler.

Filenames

An Orca program consists of several units (modules and object type definitions), each one having a specification and an implementation part. Specification parts must reside in files with names having a ‘‘.spf’’ suffix. Implementation parts must reside in files having a ‘‘.imp’’ suffix. The name of the file in which a specification part is stored must be the same as the name of the module or object type, other than for the suffix.

Calling the compiler

To compile an implementation part *prog.imp*, call

```
oc prog.imp
```

If there are no errors, this will produce the object file *prog.o*, that can be linked with other object files and the Orca run time system to create an executable file (as explained in the Orca user manual). Specification parts and generic units can not be compiled; the compiler reads them when they are needed.

Specification Parts and Generic Units

‘‘Standard’’ specification parts and generic units can be found in the directory */public/amoeba/src/util/orca/std*. When the compiler needs a specification part or generic unit, it is searched in the current directory and then in the directory mentioned above. This can be changed through a compile-time flag (see *orcamain*(U)).

Generic units are currently implemented in a preliminary way. For a generic instantiation of the form

```
OBJECT name = NEW GenericName(parameters);
```

the compiler generates two files: *name.spf* and *name.imp*. The latter file is textually included in the file containing the generic instantiation. However, if the files already exist, they are not generated again, and the body is not included in the file containing the instantiation.

See Also

ack(U), orcamain(U).
In the Programming Guide see the sections:
User Manual for the Programming Language Orca,
Orca Language Reference Manual.

Name

od − octal dump

Synopsis

```
od [-bcdhox] [file] [ [+] offset [.][b] ]
```

Description

Od dumps a file in one or more formats. If the *file* parameter is missing, *stdin* is dumped. The *offset* argument tells *od* to skip a certain number of bytes or blocks before starting. The offset is in octal bytes, unless it is followed by a ‘‘.’’ for decimal or b for blocks or both.

Options

−b Dump bytes in octal.

−c Dump bytes as ASCII characters.

−d Dump words in decimal.

−h Print addresses in hex (default is octal).

−o Dump words in octal (default).

−x Dump words in hex.

Examples

```
od -ox file
```

Dump file in octal and hex.

```
od -d file +1000
```

Dump file starting at byte 01000.

```
od -c file +10.b
```

Dumps a file starting at block 10.

Name

orcamain − Orca compiler front end

Synopsis

```
orcamain options file
```

Description

Orcamain is the Orca compiler front end. It compiles Orca implementation units into the intermediate code (EM) of the Amsterdam Compiler Kit (ACK). The compiler front end usually is invoked through the shell-script *oc*, which also feeds the EM code into the appropriate back end.

Options

The following flags are accepted by the front end:

−I*dirname*
: append *dirname* to the list of directories where specification parts and generic units are looked for. When the compiler needs a specification part or generic unit, it first looks in the current directory, then in the directories given to it by the **−I** flag in the order given, and then in the directory */public/amoeba/src/util/orca/std*.

−I
: do not look in the directory */public/amoeba/src/util/orca/std*.

−M*n*
: set maximum identifier length to *n*. The minimum value of *n* is 14, because the keyword ‘‘IMPLEMENTATION’’ is that long.

−n
: The user-declared variables will not be stored into registers on the target machine.

−w
: suppress warning messages.

Diagnostics

All warning and error messages are written on standard error output.

See Also

ack(U), oc(U).
In the Programming Guide see the sections:
User Manual for the Programming Language Orca, and *Orca Language Reference Manual*.

Name

Pascal − ACK Pascal compiler

Synopsis

```
pc  [options] files
```

Description

Pc is the Pascal front end of the Amsterdam Compiler Kit. The compiler complies as much as possible with the ISO standard proposal. The language features as processed by this compiler are described in the Pascal reference manual in the Programming Guide. Normally the compiler is called by means of the user interface program *pc*, which is a link to the ACK driver *ack*(U).

Options

−M*n* set maximum identifier length to *n*.

−n do not generate EM register messages.

−NL generate the EM *fil* and *lin* instructions that enable an interpreter to keep track of the current location in the source code.

−w suppress warning messages.

−A enable extra array bound checks, for machines that do not implement the EM ones.

−C do not map upper-case to lower-case.

−u

−U allow underscores in identifiers. It is never allowed to start an identifier with an underscore.

−a do not generate code for assertions.

−c allow C-like strings. This option is mainly intended for usage with C-functions.

−d allow the type ‘‘long.’’

−i*n* set the size of integer sets to *n*.

−s allow only standard Pascal. This disables the **−c** and **−d** options. Furthermore, assertions are not recognized at all (instead of just being skipped).

Diagnostics

All warning and error messages are written on standard error output. Descriptions of run-time errors are read from */public/module/ack/etc/pc_rt_errors*.

Files

/public/module/ack/lib/front/`arch`*/em_pem* − the Pascal front end.

See Also

libpc(L), *Pascal Reference Manual* in the Programming Guide.

Name

paste − paste multiple files together

Synopsis

```
paste [-s] [-dc] file...
```

Description

Paste displays multiple files in parallel. Suppose a set of *k* files each have one word per line. Then the *paste* output will have *k* columns, with the contents of file *j* in column *j*.

Options

−d *c* The columns are separated by tabs unless the separator is changed with the **−d** flag.

−s Print files sequentially, file k on line k. Then the first file is on line 1, the second file on line 2, etc. In effect, **−s** turns the output sideways.

Examples

```
paste file1 file2
```

Print *file1* in column 1, *file2* in column 2.

```
paste -s f1 f2 f3 f4
```

Print *f1* on line 1, *f2* on line 2, etc.

Name

pdump − dump process descriptor (checkpoint) and stack traces

Synopsis

```
pdump core-file
```

Description

Pdump expects a process descriptor (also known as checkpoint) in the *core-file*, and prints it out in symbolic form on *stderr*. Because parts of a process descriptor are architecture-dependent (especially the thread descriptors), there is a variant for each supported architecture, plus an architecture-independent variant which prints certain parts of thread descriptors as uninterpreted hex bytes.

If *core-file* is `-` then it reads from *stdin*. This feature is only intended for use by programs.

Because Amoeba executable files begin with a process descriptor, they can be dumped as well; no stack trace is printed in this case.

The program deduces which architecture the process descriptor comes from based on information in the process descriptor and attempts to analyze the thread descriptors and any stack traces accordingly.

Examples

```
pdump /bin/sh
```

might print:

```
  offset     addr+     len=     end    flags (5 segments)
000004e0 00000400+00011800=00011c00 00000110 fixed read-only text
00011ce0 00011c00+00002800=00014400 00000320 fixed read/write data
         00014400+0000edb8=000231b8 00000321 heap read/write data
         00000000+00000000=00000000 80000322 stack read/write data
000144e0 00000000+00000000=00000000 40000000 symbol table
1 threads:
TASK 0:  state    00000000
         pc            400
         sp              0
```

Assuming a checkpoint of a VAX process exists on the file */dev/dead/16*, this command might print its contents:

```
pdump /dev/dead/16
```

with possible output:

```
  offset     addr+     len=     end    flags (4 segments)
         00000400+00018800=00018c00 02000110 fixed read-only text
         00018c00+00004000=0001cc00 02000320 fixed read/write data
         0001cc00+00012c00=0002f800 02000321 heap read/write data
         7fffc000+00004000=80000000 02000322 stack read/write data
2 threads:
TASK 0: state   00000001        RUN
        pc, psl, sp      00012a8f 03c00008 0002c00c
        R0-R6   fffffffc 80050138 00000000 8025d665 00000000 00000000 003fe3ff
        R7-R13  8002148c 00000000 8005955c 0002c094 8004161c 0002c020 0002c00c
0002c00c: 00012a8f()
from      00000000
TASK 1: state   00001004        EXC
        signal        -9        Memory access control violation
        trap           8        access control violation
        pc, psl, sp      00004535 03c00009 7ffff8a0
        SPT length violation while writing 0002f800
        R0-R6   00000000 00000000 00000000 8025d665 00000000 00000000 003fe3ff
        R7-R13  8002148c 00000000 8005955c 0002f800 0002d4e4 7ffff8c4 7ffff8a8
7ffff8a8: 00004535(0)
7ffff8d0: 00004ed8(0, 1)
7ffff974: 0000c3ba()
7ffff9a8: 000044d0(2, 7ffff9d8, 7ffff9f4, 7ffffec4)
from      0000042c
```

File Formats

The input must begin with a process descriptor in network byte order; what comes after this is ignored.

See Also

ax(U), stun(U).

Name

ppload − show processor load and up time statistics

Synopsis

```
ppload [-t timeout] [-v] [directory]
```

Description

This program shows some statistics of the Amoeba (pool) processors. For each processor, it prints its name, the currently and potentially available CPU power (in MIPS), the free memory (in kbytes), and the time the processor is up.

The optional directory argument specifies an alternate directory where to look for processor names. The default is specified in *ampolicy.h* as `HOST_DIR` (typically */profile/hosts*).

Options

−t timeout Specifies the timeout (in milliseconds) used when polling a processor. The default is 2000.

−v This adds a second line for each processor, listing the version string returned by the processor.

Examples

```
ppload
```

might print:

```
mark     1.954 of 1.976 mips, 20 kb free, up 20 hrs 25 min
unlk     unreachable (proc lookup: server not found)
link     2.452 of 2.452 mips, 10552 kb free, up 6 days 4 hrs
```

An example giving verbose output:

```
ppload -v
```

which might print:

```
mark     1.954 of 1.976 mips, 20 kb free, up 20 hrs 25 min
         Amoeba 4.0 #42 by peterb@trap (00:27 01-Feb-90) in vax3500/bullet
unlk     unreachable (proc lookup: server not found)
link     2.452 of 2.452 mips, 10552 kb free, up 6 days 4 hrs
         Amoeba 4.0 #23 by jack@bitter (16:25 02-Feb-90) in sun3/workstation
```

See Also

aps(U), kstat(A).

Name

pr − print a file

Synopsis

```
pr [options] [-columns] [+page] [file] ...
```

Description

Pr formats one or more files for printing. If no files are specified, *stdin* is printed. Options are provided for setting the width and height of the page, the number of columns to use (default 1), and the page to start with, among others.

Options

−h Take next argument as page header.

−l Sets page length in lines.

−n Number the output lines.

−t Do not print page header or trailer.

−w Sets line length in characters.

Examples

```
pr -w72 -l60 file
```

Use 72 character line, 60 line page.

```
pr -3 file
```

List file three columns to a page.

```
pr +4 file
```

Start printing with page 4.

Name

prep − prepare a text file for statistical analysis

Synopsis

```
prep [file]
```

Description

Prep strips off most of the *troff/nroff* commands from a text file and then outputs all the words, one word per line, in the order they occur in the file. This file can then be sorted and compared to a dictionary (as a spelling checker), or used for statistical analysis.

Example

```
prep infile > outfile
```

Name

printenv − print out the current environment

Synopsis

```
printenv
```

Description

Printenv can be used to print out the string environment of a program, mostly the shell.

Example

```
printenv
```

This might print

```
TCPIP_SERVER=/public/cap/tcpipsvr/vmesc1
SPMASK=0xff:0x2:0x4
DISPLAY=klepper:0
ARCHITECTURE=mc68000
HOME=/home
PATH=:/home/bin:/bin:/public/util:/public/module/ack/bin/m68020
USER=versto
TERM=xterm
SHELL=/bin/sh
_WORK=/home
```

Name

put − put a capability set into a new directory entry specified by name

Synopsis

```
put [-f] pathname
```

Description

Put reads a capability set from standard input and uses *sp_append(L)* to enter it into the directory server under the specified *pathname*. Normally, the path name must not already exist in the directory server, but the use of **−f** causes any existing capset entered under that name to be replaced using *sp_replace(L).* The object(s) referred to by any existing capset are not destroyed when it is replaced.

If the input does not look like a capability set or the path name cannot be created in the directory server a suitable error message will be printed. No check for the validity of the capabilities in the set is performed.

Example

```
get /home/oldname | put /home/newname
```

will install */home/newname* in the directory server as a synonym for */home/oldname*.

See Also

chm(U), del(U), dir(U), get(U), mkd(U).

Name

pwd − print working directory

Synopsis

```
pwd
```

Description

Pwd prints the full path name of the current working directory.

Name

ref − display a C function header

Synopsis

```
ref function_name
```

Description

Ref is a program which looks up the function header of a particular function in any of a series of reference files. These reference files are produced by the *ctags*(U) program.

Ref is used by *elvis'* shift-K command.

Files

The list of files checked includes

/usr/src/lib/refs
/usr/local/lib/refs

See Also

ctags(U).

Name

refont − changes the notation used for fonts

Synopsis

```
refont [flags] files ...
```

Description

Refont reads a text file which contains font selection codes embedded within it, and it writes the same text with a different notation for fonts.

For example, the original *elvis* documentation used Epson-compatible escape sequences to select different fonts. You could use the command

```
refont -b intro.doc > intro.b
```

to make a file that uses overtyping to implement boldface or underlined text.

Options

−b Emit text which uses the ‘‘backspace’’ notation for fonts. Each underlined character will be preceded by an underscore character and a backspace character. Bold characters are set twice, with a backspace in between. The *more*(U) utility understands this notation.

−c Emit text which uses the ‘‘carriage-return’’ notation for fonts. An entire line of text is written, followed by a carriage return instead of a newline. Then a space is sent for each normal character, an underscore is sent for each underlined or italic character, and each boldface character is sent a second time. Many mainframe line printers accept this notation.

−d Emit text which uses *nroff*-style ‘‘dot’’ commands for fonts. This does not work very well.

−e Emit text using Epson-compatible escape sequences for fonts. This is useful as a ‘‘least common denominator’’ for font notations, because this is the only supported notation to use control-character sequences and also distinguish between italics and underlining.

−f Emit text which uses *nroff*’s *\fX* notation for fonts. Underlined text is denoted by *\fU*, boldface by *\fB*, italics by *\fI*, and normal text by *\fR*. This is somewhat useful in conjunction with *elvis*’ ‘‘charattr’’ option.

−x Emit text which has had all font information stripped out.

−I When reading text, **−I** tells *refont* to accept any of the above notations for fonts. Without **−I** it will ignore the ‘‘dot’’ command and *\fX* notations; they will be treated as normal text. In other words, without **−I** the only things that could be recognized as font changes are control-character sequences.

−F This causes *refont* to insert form feed characters between input files.

Warnings

Support for the *nroff*-style ‘‘dot’’ commands is not very good.

If you mix the **−I** with **−f** or **−d** flags, the resulting output may not be translatable back into its original format. This is because the original text may have included, for example, strings that looked like *\fX* strings which were not meant to cause font changes.

With **−b** or **−c**, both underlining and italics are implemented by overtyping the underscore character with a text character. Since they are represented the same way, the distinction between underlining and italics is lost.

Name

rev − reverse the characters on each line of a file

Synopsis

```
rev [file] ...
```

Description

Each file is copied to standard output with all the characters of each line reversed, last one first and first one last.

Example

```
rev file
```

Reverse each line in *file*.

Name

rm − remove a file

Synopsis

```
rm [-fir] name ...
```

Description

Rm removes one or more directory entries. If a file has no write permission, *rm* asks for permission (type ‘‘y’’ or ‘‘n’’) unless **−f** is specified.

Options

−f Forced remove: no questions asked.

−i Interactive remove: ask before removing each entry.

−r Recursively remove. If the entry refers to a directory, it will be recursively descended and remove if and only if the **−r** flag is present. The directory will also be removed once it is empty. Be careful when there are cycles in the directory hierarchy.

Examples

```
rm file
```

Remove file.

```
rm -i *.c
```

Remove all files ending in *.c*, asking about each.

See Also

del(U).

Name

rmdir − remove a directory

Synopsis

```
rmdir directory ...
```

Description

The specified directories are removed. Ordinary files are not removed.

Examples

```
rmdir /user/ast/foobar
```

Remove directory *foobar*.

```
rmdir /user/ast/f*
```

Remove all directories whose name starts with ‘‘f’’.

See Also

del(U).

Name

rush − Remote UNIX Shell

Synopsis

```
rush [-t tcpsvr] [-l user] [-i] hostname command [argument] ...
```

Description

Rush is the Amoeba equivalent of the UNIX *rsh* command. It connects the specified UNIX host using the *rsh* protocol to execute the given command. *Rush* does not provide the facilities of the UNIX *rlogin* command, it only executes one command on the remote machine.

Options

−t tcpsvr
: This option is mainly there for debugging. It specifies which TCP/IP server to use. Default: */profile/cap/tcpip/any.priv*.

−l user
: Specifies the remote user name that will be used to execute the command. The default is taken from the $USER environment variable.

−i Normally, *rush* will copy standard output and standard error from the remote command back to the local machine, but it will not copy standard input from the local machine to the remote machine. This option specifies that standard input must be copied as well.

Warnings

Since Amoeba does not have any concept of ‘‘user id’’, it is possible to impersonate any user by setting the $USER environment variable. This has the effect that once you have put an Amoeba machine in your *.rhosts* file on UNIX any Amoeba user will be able to impersonate you. This is a serious problem.

Since *rush* will pick a tcpip server at random, you should have *all* the Amoeba machines that run tcpip servers in your *.rhosts* file on UNIX. If you do not do this, random failure to connect may occur.

Example

```
rush -l ast ast.cs.vu.nl rm -r '*'
```

See Also

tcpip(A).

Name

sed − stream editor

Synopsis

```
sed [-f script_file] [edit_script] [file]
```

Description

Sed is a stream editor. It takes an edit script either from its argument or a *script_file*, and performs an edit session on a named file or *stdin*, producing output on *stdout*. With some minor exceptions, *sed* scripts in Amoeba are like those of UNIX.

Options

−f script_file
The following argument is a file containing the edit script.

Examples

```
sed -f script < file
```

Run the sed script in the file *script* on the contents of *file*.

```
sed '/pig/s//hog/g' < file
```

Replace *pig* by *hog* in *file*.

Name

session − session server for UNIX emulation (Ajax)

Synopsis

```
session [-a] [-c capname] [-d] [-f] [command [arg] ...]
```

Description

The *session server* is the central component of *Ajax*, Amoeba's UNIX emulation facility. This facility consists of the session server and a library that is linked with the application. With the help of the session server, this library implements functions like *fork* and *exec*, so a UNIX program that is not too choosy can believe it is executing in a UNIX environment. Many UNIX functions, like *read* and *write*, are implemented without the help of the session server; only functions that somehow require process management need the session server. The session server maintains a mapping from UNIX process IDs to Amoeba process capabilities, and a table that contains some process state. In separate server threads it also implements the null device (i.e., */dev/null*), pipes, and shared file descriptors referencing bullet files.

The session server provides a *personal* UNIX emulation: each user runs a separate session server, and users' session servers do not communicate. Process IDs are only unique per session server; users cannot kill each others' processes using *kill* since the processes belong to different session servers. This is somewhat analogous to the situation in a network of UNIX workstations connected by NFS, where machines share their file name space but their process ID spaces are disjoint. The difference is that in Amoeba the separation is purely logical, based on user identity, while in UNIX the separation is based on physical machine boundaries.

As a consequence, the session server does not need to maintain user IDs or group IDs per process. The access granted to a process depends solely on the capability it presents for an object; in practice this is determined by the directory graph reachable from the user's root directory.

The session server is required for the following Posix process management functions: fork(), execve() and family, wait(), sigaction(), kill(), getpid(), getppid(), getpgrp(), setpgrp(), pipe() (see *posix*(L)); by the Standard C function signal() (see *stdc*(L)); and by the Amoeba functions newproc() and newprocp() (see *newproc*(L)).

Using the session server

There are several ways to use the session server. Basically, you have a choice to specify the **−a** option or not, and you can specify a command to be run by the session server or not. (A command consists of all the arguments after the options, if any; it is interpreted as the argv argument to *exec_file*(L).)

Let us call all the processes managed by an instance of the session server a *session*. To explicitly create a new session, you start a session server without the **−a** option. If you do not specify a command, the session server will run forever and normally publishes its capability as */dev/session*. If you specify a command, a session is created to run just the command (which may spawn subprocesses that also belong to the session); the session is over (and the

session server exits) when the last process in the session finishes. In this case the session server capability is not published, unless the **−c** option is specified. To add a process to a session, the session server capability (taken from */dev/session*) should be passed as `_SESSION` in the capability environment when the process is started. *Ax*(U) does this by default if it can find */dev/session*, so normally you do not have to worry about this.

The previous mode can be described as the *manual* approach to creating sessions. The **−a** option triggers *automatic* mode. This mode preserves an existing session if one exists, but will create a new one if no (usable) session exists. If you start a session server with **−a** and no command arguments, it checks if a session server is already up and running. If this appears not to be the case, a new session is created and the session server keeps running in the background to manage it. In any case, the process that you start appears to finish as soon as it is ensured that a session exists. If you specify **−a** with a command, the command is run in the session that already existed or that was just created, and the session server appears to finish (but may really continue to run in the background) when the command finishes.

In automatic mode, all error messages that the session server may generate while serving are lost. Normally you should not worry about this (the messages are unintelligible anyway), but if you think the session server is causing problems, you should create a session manually to check it out.

(Fine point: there is some trickery going on here with processes that appear to finish while they really continue to run. What really happens is that the session manipulates its ‘owner’ property and that of the command to run so that when the command runs, the owner of the session server receives the checkpoint. When no command is run, the owner is sent a dummy checkpoint. The planned pro_swapproc should formalize these manipulations.)

Options

The session server recognizes the following options:

−a Automatic mode, see ‘using the session server’ above.

−c capname Specifies the name where the server capability is published. The default is */dev/session*, unless a command is specified and **−a** s not specified; in this case the capability is not published at all unless **−c** is given.

−d Debugging. Turns on extensive debugging output (unless the program is compiled with NDEBUG defined). The output is likely to be unintelligible to all but the author.

−f Force publishing the capability. In non-automatic mode, when the server capability must be published under a name that already exists and a server is still responding to the capability found under that name, the session server refuses to run. With **−f** specified, the capability is silently replaced in this case.

Files and Directories

/dev/session Default server capability (use **−c** option to override).

/dev/null Null device server.

/dev/proc If this directory exists, process capabilities for all running processes are stored here, with their process ID as filename. These capabilities can be presented to

stun (U) to stun them; this should only be used when the session server has crashed. When a process terminates, the session server removes its capability.

/dev/dead If this directory exists, the corpses of processes that died because of an exception or were killed by SIGQUIT are embalmed and put on display in this directory, with their process ID as filename. Such dumps can be inspected with *fromb* (U) and *pdump* (U), e.g.,

```
fromb /dev/dead/3 | pdump
```

Capability Environment

`_SESSION` The server capability is exported to the command under this name.

Limits

- The process table has space for 100 processes.
- Process IDs range from 2 to 32767 inclusive (and are reused).
- The shared file descriptor server has a maximum of 20 shared open bullet files.
- The pipe server uses two threads per open pipe; available memory and threads in the kernel may limit the number of threads that can be created.

Warnings

- Using */dev/proc* slows down process creation considerably.
- The session server prints a message when it creates a new entry in */dev/dead*. This message (like all others) is lost in automatic mode. You can use `dir -lt /dev/proc` to find out which is the latest entry.
- When the process whose process ID is used as the process group ID dies, signals can no longer be sent to the process group.
- When the session server capability is not published, */dev/null* is not published either, which causes minor grievances for the shell.

Examples

To start a permanent session server from UNIX, with its output ignored:

```
ax -n /bin/session
```

This returns as soon as the server process is started.

To start a temporary session with a shell running in it:

```
ax /bin/session /bin/sh
```

This returns when the shell returns.

See Also

ax(U), exec_file(L), fromb(U), newproc(L), pdump(U), posix(L), stdc(L).

Name

sh − shell

Synopsis

```
sh [file]
```

Description

Sh is the Amoeba command language interpreter It is known as *the shell*. It implements I/O redirection, pipes, magic characters, background processes, shell scripts and most of the other features of the V7 (Bourne) shell. A few of the more common commands are listed in the examples section, followed by a short explanation.

Examples

```
date                          # Regular command
sort <file                    # Redirect input
sort <file1 >file2            # Redirect input and output
cc file.c 2>error             # Redirect standard error
a.out >f 2>&1                 # Combine standard output and standard error
sort <file1 >>file2           # Append output to file2
sort <file1 >file2 &          # Background job
(ls -l; a.out) &              # Run two background commands sequentially
sort <file | wc               # Two-process pipeline
sort <f | uniq | wc           # Three-process pipeline
ls -l *.c                     # List all files ending in .c
ls -l [a-c]*                  # List all files beginning with a, b, or c
ls -l ?                       # List all one-character file names
ls \?                         # List the file whose name is question mark
ls '???'                      # List the file whose name is three question marks
v=/usr/ast                    # Set shell variable v
ls -l $v                      # Use shell variable v
PS1='Hi! '                    # Change the primary prompt to Hi!
PS2='More: '                  # Change the secondary prompt to More:
ls -l $HOME                   # List the home directory
echo $PATH                    # Echo the search path
if ... then ... else ... fi   # If statement
for ... do ... done           # Iterate over argument list
while ... do ... done         # Repeat while condition holds
case ... esac                 # Select clause based on condition
echo $?                       # Echo exit status of previous command
echo $$                       # Echo shell's pid
echo #                        # Echo number of parameters (shell script)
echo $2                       # Echo second parameter (shell script)
echo $*                       # Echo all parameters (shell script)
```

Name

shar − shell archiver

Synopsis

```
shar file ...
```

Description

The named files are collected together into a shell archive and written onto standard output. The individual files can be extracted by redirecting the shell archive into the shell. The advantage of *shar* over other archiving tools is that *shar* archives can be read on almost any UNIX system, whereas numerous, incompatible versions of other archiving tools are in widespread use. Extracting the files from a shell archive requires that *sed*(U) be available.

Examples

```
shar *.c > s
```

Collect C programs in shell archive *s*.

```
sh < s
```

Extract files from shell archive *s*.

See Also

sed(U).

Name

size − print text, data, and bss size of a program

Synopsis

```
size [file] ...
```

Description

Size is used to obtain information about the size of executable binary files. The text, data, bss, and total sizes for each *file* argument are printed in hexadecimal. If no arguments are present, the file name *a.out* is assumed. The amount of memory available for combined stack and data segment growth is printed in the column ‘‘stack.’’ The total amount of memory allocated to the program when it is loaded is printed in decimal under ‘‘dec.’’ This value is just the sum of the other four columns. The hexadecimal equivalent of ‘‘dec’’ is listed under ‘‘hex.’’

Example:

```
size /bin/ls
```

might print

```
text    data    bss     stack   dec     hex
e000    2000    28148   4000    213320  34148
```

and

```
size /bin/ls /bin/date
```

might print

```
text    data    bss     stack   dec     hex
e000    2000    28148   4000    213320  34148   /bin/ls
c000    2000    11ad8   4000    113368  1bad8   /bin/date
```

Name

sleep − suspend execution for a given number of seconds

Synopsis

```
sleep seconds
```

Description

The caller is suspended for the indicated number of seconds. This command is typically used in shell scripts.

Example

```
sleep 10
```

Suspend execution for 10 sec.

Name

sort − sort a file of ASCII lines

Synopsis

```
sort [-bcdfimnru] [-tx] [-o name] [+pos1] [-pos2] file ...
```

Description

Sort sorts the contents of one or more files on a per line basis. If no files are specified, *stdin* is sorted. Output is written on standard output, unless **−o** is specified. The default sort key is an entire line. The options **+pos1 −pos2** use only fields *pos1* up to, but not including *pos2* as the sort key, where a field is a string of characters delimited by spaces and tabs, unless a different field delimiter is specified with **−t**. Both *pos1* and *pos2* have the form *m.n* where *m* tells the number of fields and *n* tells the number of characters. Either *m* or *n* may be omitted.

Options

−b Skip leading blanks when making comparisons.

−c Check to see if a file is sorted.

−d Dictionary order: ignore punctuation.

−f Fold upper case onto lower case.

−i Ignore non-ASCII characters.

−m Merge presorted files.

−n Numeric sort order.

−o Next argument is output file.

−r Reverse the sort order.

−t Following character is field separator.

−u Unique mode (delete duplicate lines).

Examples

```
sort -nr file
```

Sort keys numerically, reversed.

```
sort +2 -4 file
```

Sort using fields 2 and 3 as key.

```
sort +2 -t: -o out
```

Field separator is ‘‘:’’.

```
sort +.3 -.6
```

Characters 3 through 5 form the key.

Name

split − split a large file into several smaller files

Synopsis

```
split [-n] [file [prefix]]
```

Description

Split reads *file* and writes it out in *n*-line pieces. By default, the pieces are called *xaa*, *xab*, etc. The optional *prefix* argument can be used to provide an alternative prefix for the output file names.

Options

−n *n* specifies the number of lines per piece (default: 1000).

Examples

```
split -200 file
```

Split file into pieces of 200 lines each.

```
split file z
```

Split file into *zaa*, *zab*, etc.

Name

std_copy − replicate an object

Synopsis

```
std_copy servercap original newcopy
```

Description

As part of the automatic replication facilities, servers implementing abstract objects (such as files) support the STD_COPY command which requests them to replicate a particular object. This allows a higher degree of availability for an object by storing copies of it on several different servers. The automatic replication facility of the Soap Server is implemented using the STD_COPY command and is the preferred method of replication. However, in some cases it is convenient to be able to execute the command for a particular object. *std_copy* is a utility which permits the user to directly execute an STD_COPY command. *servercap* is the path name for the capability for the server which is to make the copy, *original* is the path name for the capability for the object to be copied and *newcopy* is the path name to store the capability for the new copy of the object.

This differs from the standard copy command in that it works for all abstract objects and not just files and, more importantly, it allows you to specify a particular server on which to make the copy.

Diagnostics

Three types of error are possible:

1) Errors while looking up the capabilities of the server and the original object in the directory server.
2) Errors during the actual copy operation itself (for example, insufficient permission to copy the object).
3) Errors while attempting to store the capability of the newly copied object in the directory server.

The error messages for the three different types are clear and include the explanation of the error status.

Warnings

In most cases there is little sense in replicating on the same server as the original object. To increase the availability of the file it is best to select a different server from the one storing the original.

Example

```
std_copy /public/cap/bulletsvr/b2 /home/myfile /home/myfile2
```

makes a copy of */home/mydata* on the bullet file server known as *b2* and stores the capability for the copy under the name */home/myfile2*

See Also

cp(U), soap(A), std_destroy(U), std_info(U), std_restrict(U), std_touch(U).

Name

std_destroy − destroy an object

Synopsis

```
std_destroy objectcap ...
```

Description

Std_destroy executes an `STD_DESTROY` command for each object in its argument list. If the destroy operation succeeds it also deletes the directory entry for the object. Only servers for abstract objects accept this command. (Servers for physical objects, such as a disk, would need special assistance to implement it.) Most servers require a special right to be set in the capability to execute this command.

The exit status of the command is the number of errors encountered.

Diagnostics

Three types of error are possible:

1) Errors while looking up the capability of the object in the directory server.
2) Errors during the actual destroy operation itself (such as insufficient permission).
3) Errors while attempting to delete the capability for the object from the directory server.

The error messages for the three different types are clear and include the explanation of the error status.

Example

```
std_destroy /home/myfile /home/bin
```

will destroy the two objects */home/myfile* and */home/bin* and delete their capabilities from the directory server.

See Also

rm(U), del(U), std_copy(U), std_info(U), std_restrict(U), std_touch(U).

Name

std_info − get the standard information about an object

Synopsis

```
std_info capability
```

Description

Std_info prints the standard information string for the object whose capability is *capability*. This information string is the same as printed by the

```
dir -l
```

command.

Several servers have highly abbreviated information strings. The following lists information strings for well known objects:

```
/       Soap directory
%       Soap kernel
-       Bullet file
@       Virtual disk
+       Serial port
```

In addition to the character further information such as the size of the object or the rights are also printed. For more details about the standard information for a server see the manual page for the server.

Diagnostics

Two types of error are possible:

1) Errors while looking up the capability of the object in the directory server.
2) Errors during the actual standard info operation itself.

The error messages for the three different types are clear and include the explanation of the error status.

Example

```
std_info /home/pclient
```

where */home/pclient* is a file produces

```
-    41184 bytes
```

See Also

dir(U), std_copy(U), std_destroy(U), std_restrict(U), std_touch(U).

Name

std_restrict − produce a version of a capability with restricted rights

Synopsis

```
std_restrict original mask newcap
```

Description

Std_restrict asks the server responsible for the capability *original* to produce a version of the capability but with on the bits set in *mask* still set in the rights field of the capability. It cannot add rights. It can only remove them. The resultant capability is stored under the name *newcap*. The *mask* is interpreted as a hexadecimal number.

Diagnostics

Three types of error are possible:

1) Errors while looking up the original capability in the directory server.
2) Errors during the actual restrict operation itself.
3) Errors while attempting to store the restricted capability in the directory server.

The error messages for the three different types are clear and include the explanation of the error status.

Example

```
std_restrict /home/mine 1f /home/mine.public
```

will produce a capability for the same object as */home/mine* but with only the rights in the bottom 5 bits switched on that were also switched on in */home/mine*. The capability will be stored in */home/mine.public*.

See Also

rm(U), dir(U), std_copy(U), std_destroy(U), std_restrict(U), std_touch(U).

Name

std_status − get server status

Synopsis

```
std_status server
```

Description

Std_status prints the status information for the server whose capability is given. This is usually a couple of lines up to a couple of pages of output. The layout of the server status is documented with each individual server.

Diagnostics

Error messages are self-explanatory.

Example

```
std_status /profile/pool/mc68000/.run
```

might print

```
HOST     STAT SINCE LREPL LPOLL NPOLL NBEST NCONS   MIPS   NRUN  MBYTE
unlk     DOWN 45:02   54s   44s  7460     5     5
bpl      UP   43:49    0s    0s  3577    76   990  0.900  0.000  2.524
bmi      UP   49:06    3s    3s  7403   114   990  0.900  0.000  2.481
blo      UP   40:25    1s    1s 29110    48   774  0.900  0.000  3.119
```

See Also

bstatus(A), runctl(A), std_info(U), std_status(L).

Name

std_touch − touch an object so that the server does not garbage collect it

Synopsis

```
std_touch objectcap ...
```

Description

Std_touch executes an `STD_TOUCH` command for each object in its argument list. The purpose of this is to tell the server not to garbage collect the object. (If the capability for an object is lost there is no way to recover the resources it is using other than to garbage collect all objects not accessed within a certain time period.) The exit status of the command is equal to the number of errors encountered.

Diagnostics

Two types of error are possible:

1) Errors while looking up the capability of the object in the directory server.

2) Errors during the actual touch operation itself (such as insufficient permission).

The error messages for the three different types are clear and include the explanation of the error status.

Example

```
std_touch /home/myfile
```

will tell the Bullet Server that */home/myfile* should not be garbage collected since someone still has an interest in it.

See Also

std_age(A), std_copy(U), std_destroy(U), std_info(U), std_restrict(U).

Name

strings − print all the strings in a binary file

Synopsis

```
strings file ...
```

Description

Strings looks for sequences of ASCII characters followed by a zero byte. These are usually strings. This program is typically used to help identify unknown binary programs.

Options

− search the whole file, not just the data segment.

−o print octal offset of each string.

−n *n* is minimum length string (default = 4).

Examples

```
strings -5 a.out
```

Print the strings longer than 4 characters in *a.out*.

```
strings - /bin/sh
```

Search entire shell file (text and data).

Name

stun − stun (paralyze, possibly kill) a process

Synopsis

```
stun [-d dumpfile] [-m host] [-s stun-code] [-p] [-c] [-v] process ...
```

Description

This program sends a ‘‘stun’’ request to the specified processes. Processes are named by their capability. For example, */process/sh*, */dev/proc/2* or perhaps */profile/hosts/bcd/ps/1*. What happens when a process receives a stun request that depends on its owner? If there is no owner or if the owner does not care, the process is destroyed.

Options

−d dumpfile
: Specifies that the core file should have the name *dumpfile*. The default it that it is put in */dev/dead/$$*, where $$ is the process id of the process stunned.

−m host
: Causes the *process* argument(s) to be interpreted as process identifiers in the directory */public/hosts/*`host`*/ps*.

−s stun-code
: Set the stun code. Some owners look at the stun code sent along with the stun request; For example, the session server reports the stun code to a process’ parent as the signal that killed it. The default is 3, or `SIGQUIT`.

−c Print a traceback and continue: stun temporarily takes over ownership of the process, prints a traceback (see *pdump*(U)), and lets the process continue; the original owner is restored but never notified.

−p Print a traceback: stun temporarily takes over ownership of the process, prints a traceback (see *pdump*(U)), and then passes the checkpoint on to the original owner.

−v Verbose: print comments about what the program is doing.

Warning

The use of the **−c** option is not completely transparent to the process being stunned; *trans*, *getreq* and *mu_trylock* calls may fail causing it to report an error and die.

Examples

```
stun /process/zip
```

To stun a process, see a checkpoint, and see what stun is doing:

```
stun -v -p /process/zap
```

See Also

pdump(U).

Name

sty − show or change tty settings

Synopsis

```
sty [tty <cap>] [all]
sty [+-]help
sty [tty <cap>] [reset | size]
[[+-]<flag>] ... [[+-]<char>] ... [<char> <value>] ...
```

Description

A tty controls the way the keystrokes you type to a process are interpreted, and the way read and write transactions of the process are handled. The tty interface keeps flags and characters, which can be displayed and set by *sty*. The flags and characters that *sty* understands are defined by Posix 1003.1. For each flag and character, *sty* knows a default value which is suitable for use on Amoeba. Without any options *sty* prints the current tty settings, insofar as they differ from the compiled-in defaults.

Options

all
: makes *sty* print each and every flag and character, instead of just the ones that differ from the defaults.

[+-]help
: tells you how to use *sty*. **-help** gives a very short synopsis, **help** also tells what the defaults for the flags and characters are, and finally **+help** lists the meaning of each and every flag and character.

tty <capability>
: tells *sty* to use capability as the capability for the tty instead of the default `STDIN` from the capability environment.

reset
: means use the compiled-in defaults instead of the current tty settings.

size
: prints the size of the tty in rows and columns of characters, if known.

To turn off a flag, pass the option −<*flagname*>. To turn on a flag, pass the option <*flagname*>. To set a flag to its default, pass the option +<*flagname*>.

To disable a ttychar, pass the option −<*charname*>. To set a ttychar, pass the option <*charname*> <*charvalue*>, where <*charvalue*> is of the form *0x*<*hexvalue*>, ^<*character*> or <*character*>. To set a ttychar to its default, pass the option +<*charname*>. (Note that character ‘‘^’’ is special to the shell, so it needs to be quoted or preceded by a backslash.)

The output of *sty* is usable as arguments to *sty* itself, so you can save your tty settings in a readable form with

```
sty > NiceSettings
```

and then later restore them with

```
sty ‘cat NiceSettings‘
```

For a complete list of characters and flags, please consult the Posix document, or type

```
sty +help
```

Environment

If no tty option is passed, *sty* uses the capability `STDIN` as tty capability.

Warning

Ax attempts to emulate a Posix tty interface on top of your native UNIX tty interface, which cannot be done 100% correctly.

Examples

Here we show three ways to set the ttychar ‘‘eof’’ to CTRL−D:

```
sty +eof
```

works because CTRL−D is the default for this char.

```
sty eof 0x4
```

works because 0x4 happens to be CTRL−D.

```
sty eof ’^d’
```

is the common way to set eof to CTRL−D.

To turn off the possibility to signal eof at all, type:

```
sty -eof
```

To display the complete tty configuration, type:

```
sty all
```

If all settings have their default values, this produces:

```
brkint icrnl -ignbrk -igncr -ignpar -inlcr -inpck -istrip ixoff ixon
-parmrk opost -clocal cread -cstopb -hupcl -parenb -parenodd echo echoe
echok -echol icanon -iexten isig -noflsh -tostop eof ’^D’ -eol erase ’^H’
intr ’^C’ kill ’^X’ quit ’^´ -susp start ’^Q’ stop ’^S’ min 1 time 0
baud 9600
```

See Also

Posix Standard 1003.1-1988

aterm(U), ax(U), tios_getattr(L), tios_setattr(L), tios_getwsize(L).

Name

sum − compute the checksum and block count of a file

Synopsis

```
sum file
```

Description

Sum computes the checksum of one or more files. It is most often used to see if a file copied from another machine has been correctly received. This works best when both machines use the same checksum algorithm.

Examples

```
sum /public/users/ast/xyz
```

might print

```
38349   161
```

Name

tail − print the last few lines of a file

Synopsis

```
tail [-n] [file]
```

Description

The last few lines of one file are printed. The default count is 10 lines. The default file is *stdin*. If more than one *file* argument is given only the first is used. The rest are ignored.

Options

−n Says to print the last *n* lines of the input.

Examples

```
tail -6
```

Print last 6 lines of *stdin*.

```
tail -1 file1 file2
```

Print last line of *file1*. The second file, *file2*, is ignored.

Name

Tape Server − a server interface to tape controllers

Synopsis

This is a server currently built into the kernel.

Description

A tape is a record oriented backup/restore device. The smallest addressable unit is a record. The record size can vary from 1 byte to MAXINT bytes, depending on tape hardware and host operating system (due to the maximum transaction buffer size of Amoeba, a maximum record size of 29696 bytes is used).

The end of a sequence of records can be marked by an ‘‘End-Of-File’’ marker. This marker is a special record which can be used to define tape files of records. Some tape controllers are able to search for EOF markers.

The tape server provides a general interface to tape devices. Clients use the tape server to control tape devices. The tape server uses internal tape drivers to access the tape devices.

A tape device consists of one or more tape units. A tape unit is identified by a capability.

The tape server consists of two parts: the tape server itself and the internal tape drivers. The following section deals with the tape server interface, followed by sections concerning tape drivers.

Tape server interface

Each tape operation handled by the tape server requires a tape capability, This capability can be found in the tape unit’s host machine directory. The capability is always formed as follows:

```
/profile/hosts/<machine>/tape:<number>
```

The <*number*> parameter specifies which internal driver and tape unit is concerned.

The tape server recognizes a set of predefined commands. All commands return an error code of the operation, which can be one of the standard errors, Amoeba errors or tape I/O errors.

The returned standard errors are predefined constants (see *stderr*(L)). `STD_COMBAD` is returned when not using the predefined tape commands, `STD_CAPBAD` is returned when using bad capabilities and `STD_ARGBAD` is returned when passing bad arguments to tape stubs (i.e., a record size which is too big). Upon a successful operation `STD_OK` is returned.

Since tape actions involve Amoeba transactions, transaction errors might be returned (see *trans*(L)).

Following the possible tape I/O error codes with their explanation. **The reader should be aware that several error codes are obsolete, in the near future they will be changed to more generalized error codes.**

TS_INVALID_CMD	An invalid command has been issued to the tape controller (tape driver error, obsolete).
TS_CMD_ABORTED	The last command has been aborted (obsolete).

TS_UNIT_OFFLINE	The tape unit is off-line.
TS_UNIT_AVAILABLE	The tape unit is available; no tape has been loaded (obsolete, will be TS_UNIT_OFFLINE).
TS_MEDIA_FMT_ERR	Tape format error.
TS_WRITE_PROT	The tape is write protected.
TS_CMP_ERR	Cannot write data to tape.
TS_DATA_ERR	Invalid record on tape.
TS_HOST_BUF_ACCESS_ERR	Controller cannot access user buffer (tape driver error, obsolete).
TS_CONTR_ERR	Controller error.
TS_DRIVE_ERR	Drive error.
TS_FRMT_ERR	Format error.
TS_BOT_ERR	While repositioning, hit begin of tape marker.
TS_TAPE_MARK	Found an EOF marker.
TS_REC_DAT_TRUNC	Tape record longer than expected.
TS_POS_LOST	Tape record position lost.
TS_SER_EXCP	Serious exception, a double fault.
TS_OTHER	Some other tape error.

The following summary specifies which operations are valid for the tape server, along with a short description. For the interface specification, see *tape*(L). Along with the tape commands (commands which are prefixed by tape_), the tape server recognizes standard commands (commands which are prefixed by std_, see *std*(L)).

Currently, rights are not implemented in the tape server. Everybody has full access to the tape server.

The interface is also described in the library sections of the manual. (see *tape*(L) *std*(L)).

std_info	At a std_info call, the tape server request the tape driver an identification of a tape unit. The tape server returns a null terminated, formatted ASCII string.
tape_init	Tape_init initializes a tape unit and also loads an inserted tape.
tape_read	Tape_read reads a tape record from a tape.
tape_write	Tape_write writes a tape record onto a tape.
tape_write_eof	Tape_write_eof writes an `EOF` marker onto the tape.
tape_rewind	Tape_rewind rewinds a tape to begin of tape. Users can specify to rewind the tape synchronously or a-synchronously. When synchronous, tape_rewind finishes once the tape is completely rewound. In the latter case, the command finishes directly; there is not waited until the tape is actually rewound.
tape_status	Tape_status requests the status of a tape unit. Tape_status returns a null terminated formatted ASCII string which contains the complete status of a unit. The format is tape driver dependent. For a

description of the contents, see the tape driver sections.

tape_rskip — Tape_rskip skips a number of records on tape. Negative skip counts result in backward skipping.

tape_fskip — Tape_fskip skips a number of EOF markers on tape. Negative counts result in backwards skipping. Forward skipping causes the new tape position to be at the beginning of the file, negative skips causes the new tape position to be at the EOF marker.

tape_erase — Tape_erase erases a tape. It formats every inch on tape and since tapes can be very long and slow, this command may take some time to finish.

tape_setlvl — Tape_setlvl is for debugging tape drivers only. It is used for setting some debug output level in the driver.

tape_why — Tape_why converts an Amoeba or tape error code to a null terminated ASCII string (in fact it returns a pointer to an internally kept error table). This routine doesn't require a capability.

TMSCP tape driver

The tmscp driver is a tape server internal driver which is linked to the tape server module. Users can use the tmscp driver only via the tape server.

The tmscp tape driver is a driver which controls Q22 bus tmscp tape controllers. These controllers are able to manage DEC TK50/TK70 tape units. For each of the above described tape server commands, associated driver routines are implemented (i.e., TS_REC_SKIP, tmscp_record_skip). Each driver routine returns a tape error code as described above or standard errors (see *stderr*(L)). Upon a successful operation STD_OK is returned.

The tape driver defines three basic tape units states. At initialization the tape unit is marked **UNIT_INIT.** During this state the tape driver is not accessible. Once it has passed the UNIT_INIT state, it becomes available (**UNIT_AVAILABLE**). In this state the controller is initialized and the driver tries to load a tape. If no tape is inserted in the cabinet, the unit stays available. Once it has loaded the tape, the unit is marked **UNIT_ONLINE.** In this state it is accessible for normal tape read/write actions. To mark a tape on-line while all is up and running, the unit must be re-initialized.

In addition to the three basic states the following states are defined:

UNIT_WRITE_PROT — The tape unit is write protected (it is possible tape units are write protected, but I don't know how).

UNIT_BUSY — The tape unit is busy doing some command.

UNIT_FILE_POS — Due to some record skips, the file position is lost (explained below).

UNIT_REC_POS — Due to some illegal tape movement, the controller has lost the record position.

Apart from the above described status, the following values are kept per unit.

file_pos — The current file position on tape. Each time the controller finds an EOF marker, this number is increased. It is reset when begin of tape is found. When UNIT_FILE_POS is set this value might be wrong (explained below).

record_pos	The current record position on tape. This value is reported by the tape controller at each tape I/O action. When UNIT_REC_POS is set this value might be wrong.
last_command	The last command issued to the tape controller. When UNIT_BUSY is set, the controller is still busy doing the command.
format	A value informing which type of tape is mounted (currently only RV80, TK50 and 9-track devices are recognized).
last_fail	Last failure holds the error code found when executing last_command.

This data can be fetched from the driver via the tape server TS_STATUS command. The tape server fetches the data by calling the tmscp driver status routine, which will format the data in a printable ASCII string. Following, typical output produced by tmscp_status:

```
tmscp 0:

Unit_status:          Online.Begin_of_tape
Tape_file_pos:        000000
Tape_record_pos:      000000
Last_command:         Set unit characteristics
Last_failure:         None
Controller_status:    Running
Miscellaneous:
Unit type:            TK50 compatible unit
Format type:          High density
```

In this case, tmscp unit 0 is on-line, and at begin of tape. The tape file and record position are zero. The last command was a "Set unit characteristics" and did not fail. The controller is up and running. The tape itself is an high density TK50 compatible tape.

```
tmscp 0:

Unit_status:          Online
Tape_file_pos:        000000?
Tape_record_pos:      000910
Last_command:         Repos
Last_failure:         Data error
Controller_status:    Running
Miscellaneous:
Unit type:            TK50 compatible unit
Format type:          High density
```

In this case, the unit is on-line. The tape record position is record number 910, the file position is uncertain. The last command was a repositioning command, which failed on a data error.

As previously said, tapes are record based. The end of a tape record sequence is marked by an EOF marker. The tmscp controller is able to skip forward or backward to EOF markers. When skipping forward to EOF markers, the new tape position will be the beginning of the desired file. When skipping backward, the new tape position will be the end of the desired

file (actually at the EOF record). To skip back one file, better skip back two files and skip one record forward.

When skipping records on a tape, the tmscp controller cannot see the difference between data and EOF records. In this case it is possible EOF markers are skipped without noticing, so file positions are insecure after skipping records. The UNIT_FILE_POS flag will be set after each record skip. The flag is cleared once the driver notices begin of tape.

See Also

std(L), stderr(L), tape(L), trans(L).

Name

tar − tape archiver

Synopsis

```
tar [cxtvoF] tarfile file ...
```

Description

Tar is an archiver in the style of the standard tape archiver. It does not have to write the archive to tape but can write it to any file. Its primary advantage over *ar* is that the *tar* format is somewhat more standardized than the *ar* format, making it theoretically possible to transport binary files to another computer, but do not bet on it.

Options

−F Force tar to continue after an error.

−c Create a new archive.

−t Print a table listing the archive's contents.

−v Verbose mode − tell what is going on as it happens.

−x The named files are extracted from the archive.

Examples

```
tar c /dev/fd1 file1 file2
```

Create a two-file archive.

```
tar xv /dev/fd1 file1 file2
```

Extract two files from the archive.

```
(cd src; tar c -) | (cd dest; tar x -)
```

Copy the directory tree with root *src* to *dest*.

Name

tee − divert *stdin* to a file

Synopsis

```
tee [-ai] file ...
```

Description

Tee copies its standard input to its standard output. It also makes copies on all the files listed as arguments.

Options

−a Append to the files, rather than overwriting them.

−i Ignore interrupts.

Examples

```
cat file1 file2 | tee xxx
```

Save the contents of the two files *file1* and *file2*. in *xxx* and also display it on the standard output.

```
pr file | tee x | lpr
```

Save the output of *pr* on *x* while also sending it to the program *lpr*.

Name

tell − send message to user or topic of user-to-user message service

Synopsis

```
tell [-n] ( -u user | -t topic | user-or-topic ) [text] ...
```

Description

Tell is the input half of a simple user interface to Amoeba's user-to-user message services. It can send a message directly to another user or to a group of users interested in a particular ''topic.'' The message text is taken from the command line or read from standard input; the message is normally prefixed with a header indicating the sender and the time it was sent.

Users listen for incoming messages with the *listen*(U) server. They manage the list of topics in which they are interested with (un)subscribe, see *topics*(L). The actual message transfer is mediated by the multicast server, see *mc*(A) and *mc*(L).

Options

The following options are recognized:

−n	Suppresses the header and trailer that are normally put around a message. As a special case, if an empty message body is given with **−n**, a zero-length message is sent, which is ignored by listeners. This can be used to poll a topic or user without actually producing output at the listeners.
−u user	Specifies a user name to send the message to.
−t topic	Specifies a topic name to send the message to.
user-or-topic	If neither **−t** nor **−u** is specified, the first argument after the options is used as the destination; this can be either a user name or a topic name (but it must be unambiguous).
text	If more arguments follow, a one-line message is constructed from them, prefixed with a short header unless **−n** is given.

If no text arguments are given, a multi-line message is read from standard input. In this case, unless suppressed by **−n**, the header is longer, on a line by itself, and a line reading ''EOF'' follows the message. When input is interactive (from a tty device), each message is sent as a separate line, so two users can engage in a conversation by each using *tell* with the other's name as destination.

Diagnostics

Send complains when it cannot find a specified user or topic, when the destination is ambiguous, or when the actual message transfer fails.

When a user has no running listen servers, or no users interested in a topic have a running listen server, the message

```
tell: no active 'listen' servers for 'username'
```

is printed.

When attempting to send to a user who has never run a listen server, the message

```
tell: user 'username' never started a 'listen' server
```

appears.

Environment Variables

The string environment variables `LOGNAME` and `USER` are tried (in that order) to determine the user name of the originator for the header.

Files

profile/module/ms/`topic` − is the multicast group for *topic*.

/profile/users/`user`*/profile/ms/relay* − is the multicast group for *user*.

Warnings

You must have a directory */profile/users* to send to users. This should be a link to */public/users*. The command

```
get /public/users | put /profile/users
```

will accomplish this.

When sending to a user or topic for which one of the listeners has recently crashed, a 5-seconds delay may occur for each deceased listener as the multicast server attempts to contact the locate listener.

Example

To send a simple message to user Joe, try:

```
tell joe Are you there?
```

See Also

listen(U), mc(A), mc(L), topics(U).

Name

termcap − print the current termcap entry

Synopsis

```
termcap [type]
```

Description

Termcap reads the file */etc/termcap* to obtain the entry corresponding to the terminal type supplied as the argument *type*. If none is given, the current value of the environment variable `TERM` is used. It then prints out all the parameters that apply.

Example

```
termcap
```

Print the termcap entry for the current terminal which might look like

```
TERM = sun

End of line wraps to next line   (am)
Ctrl/H performs a backspace      (bs)
Number of columns                (co) = 80
Number of lines                  (li) = 34
Clear to end of line             (ce) = ^[[K
Clear to end of screen           (cd) = ^[[J
Clear the whole screen           (cl) = ^L
Start "stand out" mode           (so) = ^[[7m
End "stand out" mode             (se) = ^[[m
Start underscore mode            (us) = ^[[4m
End underscore mode              (ue) = ^[[m
Start bold mode                  (md) = ^[[1m
Start reverse mode               (mr) = ^[[7m
Return to normal mode            (me) = ^[[m
Cursor motion                    (cm) = ^[[%i%d;%dH
Up one line                      (up) = ^[[A
Right one space                  (nd) = ^[[C
Generated by "UP"                (ku) = ^[[A
Generated by "DOWN"              (kd) = ^[[B
Generated by "LEFT"              (kl) = ^[[D
Generated by "RIGHT"             (kr) = ^[[C
Generated by "PGUP"              (k1) = ^[[224z
Generated by "PGDN"              (k2) = ^[[225z
Generated by numeric "+"         (k3) = ^[[226z
Generated by numeric "-"         (k4) = ^[[227z
Generated by numeric "5"         (k5) = ^[[228z
```

Name

test − test for a condition

Synopsis

```
test expr
[ expr ]
```

Description

Test checks to see if files exist, are readable, etc. and returns an exit status of zero if true and nonzero if false. The legal operators are

−r file	true if the file is readable
−w file	true if the file is writable
−x file	true if the file is executable
−f file	true if the file is not a directory
−d file	true if the file is a directory
−s file	true if the file exists and has a size > 0
−t fd	true if file descriptor *fd* (default 1) is a terminal
−z s	true if the string *s* has zero length
−n s	true if the string *s* has nonzero length
s1 **=** *s2*	true if the strings *s1* and *s2* are identical
s1 **!=** *s2*	true if the strings *s1* and *s2* are different
m **−eq** *n*	true if the integers *m* and *n* are numerically equal

The operators **−gt**, **−ge**, **−ne**, **−le** and **−lt** may be used as well in place of **−eq**. These operands may be combined with **−a** (Boolean and), **−o** (Boolean or), **!** (negation). The priority of **−a** is higher than that of **−o**. Parentheses are permitted, but must be escaped to keep the shell from trying to interpret them.

Example

```
test -r file
```

See if *file* is readable.

```
[ -r file ]
```

The same but with the other syntax.

Name

time − report how long a command takes

Synopsis

```
time command
```

Description

The command is executed and the real time, user time, and system time (in hours, minutes, and seconds) are printed.

Examples

```
time a.out
```

Report how long *a.out* takes. It might look like this:

```
real       16.0
user        0.0
sys         0.0
```

(UNIX ONLY)

Name

tob − copy a UNIX file to an Amoeba Bullet Server

Synopsis

```
tob [-n] unix-name amoeba-name [fileserver]
```

Description

Tob is a command which runs only under UNIX with an Amoeba driver. It reads the file *unix-name* from a UNIX system and stores it on an Amoeba Bullet Server under the name *amoeba-name* in the Amoeba directory service. This name must not exist already.

The optional third parameter allows you to specify the path name in the Amoeba directory server for the capability of a particular Bullet Server to use. Otherwise the default Bullet Server is used.

Options

The **−n** option requests that the capabilities for the bullet file and optional fileserver not be stored or looked up in the Amoeba directory server. Instead they should be stored or looked up in UNIX files with the specified name(s). This feature is for bootstrapping a system when no directory server is available.

Diagnostics

These are all self explanatory.

Warnings

The UNIX file must fit in core on the UNIX host since *tob* tries to send the file as a single create command.

Example

```
tob /etc/termcap /home/mytermcap
```

will copy the UNIX file */etc/termcap* to the default Bullet Server and store under the name */home/mytermcap* under Amoeba.

See Also

bullet(A), fromb(U).

Name

topics − manage multicast topic group

Synopsis

```
subscribe [topic ...]
unsubscribe topic ...

newtopic topic ...
deltopic topic ...
```

Description

These programs manage ‘‘topic’’ multicast groups, used by *tell* (U) and *listen* (U).

Subscribe adds the given topics to your interest list; you will receive all messages sent to those topics. *Subscribe* with no options lists the existing topics in alphabetical order; topics to which you are subscribed are prefixed with an asterisk.

Unsubscribe removes the given topics from your interest list; you will no longer receive any messages sent to those topics.

Newtopic creates a new topic, to which users may subscribe.

Deltopic deletes a topic list and all its subscriptions.

Files and Directories

/profile/cap/mc/mc − is the capability for the multicast server, used to create new topics.

/profile/module/ms/`topic` − is the multicast group for *topic*.

/profile/ms/relay − is the multicast group for the current user.

/profile/ms/`topic` − is a dummy file created to indicate subscription to *topic*.

Warnings

After deleting a topic and creating a new topic with the same name, users who were subscribed will still appear to be subscribed, but will not receive messages sent to the topic.

These programs are shell scripts, and are necessarily more fragile than programs with the same functionality.

Examples

To list the topics currently available:

```
subscribe
```

This might print:

```
* amoeba
* backup
  coffee
  daisy
* test
  views
```

To create a new topic ‘‘beer’’:

```
newtopic beer
```

To start receiving messages on topic ‘‘beer’’:

```
subscribe beer
```

To stop receiving test messages:

```
unsubscribe test
```

The topic list now looks like this:

```
* amoeba
* backup
* beer
  coffee
  daisy
  test
  views
```

See Also

dir(L), listen(U), mc(A), mc(L), tell(U).

Name

tr − translate character codes

Synopsis

```
tr [-cds] [string1] [string2]
```

Description

Tr performs simple character translation. When no flag is specified, each character in *string1* is mapped onto the corresponding character in *string2*.

A string may consist of one or more of

range e.g.: [a-z] or [0-9]

escape e.g.: \012 (meaning newline) or \\ (‘‘\’’ itself)

other e.g.: A or (or 1

Options

−c Complement the set of characters in string1.

−d Delete all characters specified in string1.

−s Squeeze all runs of characters in string1 to one character.

Examples

```
tr ’[a-z][A-Z]’ ’[A-Z][a-z]’ < x > y
```

Convert upper case to lower case, and vice versa, reading the input from the file *x* and writing it to the file *y*.

```
tr -d ’0123456789’ < f1 > f2
```

Write on the file *f2* the contents of *f1* but with all the digits deleted.

Name

true − exit with the value true

Synopsis

```
true
```

Description

This command returns the value true (i.e. 0). It is used for shell programming.

Example

```
while true
do
        ls -l
done
```

List the directory until interrupted.

Name

tset − set the TERM string environment variable

Synopsis

```
tset [device]
```

Description

Tset is used almost exclusively to set the shell variable `TERM` from inside profiles. If an argument is supplied, that is used as the value of `TERM`. Otherwise it looks in */etc/ttytype*.

Example

```
eval ‘tset‘
```

Set `TERM`.

Name

tsort − topological sort

Synopsis

```
tsort file
```

Description

Tsort accepts a file of lines containing ordered pairs and builds a total ordering from the partial orderings.

Example

```
ar cr libc.a ‘lorder *.s | tsort‘
```

Build library.

Name

tty − print the device name of this tty

Synopsis

```
tty
```

Description

Print the name of the controlling tty.

Options

−s Silent mode (return status only).

Example

```
tty
```

Print the tty name.

Name

unexpand − convert spaces to tabs

Synopsis

```
unexpand [-a]
```

Description

Unexpand replaces spaces in the named files with tabs. If no files are listed, *stdin* is given.

Options

−a The **−a** flag is used to force all sequences of spaces to be expanded, instead of just leading spaces (the default).

Example

```
unexpand oldfile > newfile
```

Convert leading spaces in *oldfile* to tabs and save the result in *newfile*.

Name

uniq − delete consecutive identical lines in a file

Synopsis

```
uniq [-cdu] [+n] [-n] [input [output]]
```

Description

Uniq examines a file for consecutive lines that are identical. All but duplicate entries are deleted, and the file is written to output.

Options

−c Give count of identical lines in the input.

−d Only duplicate lines are written to output.

+n This option skips the first *n* fields, where a field is defined as a run of characters separated by white space.

−n This option skips the first *n* spaces. Fields are skipped first.

−u Only unique lines are written to output.

Examples

```
uniq +2 file
```

Ignore first 2 fields when comparing.

```
uniq -d inf outf
```

Write duplicate lines from the file *inf* to *outf*.

See Also

sort(U).

Name

uud − decode a binary file encoded with uue

Synopsis

```
uud [-n] [-s srcdir/] [-n dstdir/] file
```

Description

Uud decodes a file encoded with *uue* (U) or UNIX uuencode. The decoded file is given the name that the original file had. The name information is part of the encoded file. Mail headers and other junk before the encoded file are skipped.

Options

−n Do not verify checksums.

−s Name of directory where *.uue* file is.

−t Name of directory where output goes.

Examples

```
uud file.uue
```

Re-create the original file.

```
uud - < file.uue
```

The ‘‘−’’ means use *stdin*.

See Also

uue(U).

Name

uue − encode a binary file to ASCII (e.g., for mailing)

Synopsis

```
uue [-n] file [-]
```

Description

Uuencode is a famous program that converts an arbitrary (usually binary) file to an encoding using only 64 ASCII characters. Uudecode converts it back to the original file. The *uue* and *uud*(U) programs are enhanced versions of these programs, and are compatible with the UNIX ones. The files produced can even be sent successfully over BITNET, which is notorious for mangling files.

It is possible to have *uue* automatically split the encoded file up into chunks small enough to fit into mail messages. The output files then get the suffixes *.uaa*, *.uab*, etc., instead of *.uue*. When *uud* is given *file.uaa* to decode, it automatically includes the subsequent pieces. The encoding takes 3 bytes (24 bits) from the input file and renders it as 4 bytes in the output file.

Options

−n *n* is a decimal integer specifying how many lines to put in each output file.

Examples

```
uue file
```

Encode *file* to *file.uue*.

```
uue file - > x
```

Encode *file* and write the result on *stdout*.

```
uue -800 file
```

Output on *file.uaa*, *file.uab* etc.

See Also

uud(U).

Name

virec − recover the modified version of a file after a crash

Synopsis

```
virec [-d tmpdir] textfile ...
virec < /tmp/elvxxxx
```

Description

When you edit a file with *elvis* (U), only about 5K bytes of the file are stored in RAM; the rest is stored in a file in */tmp*. This allows you to edit files larger than a process' data space. It also plays a big part in the way that ''undo'' and ''paste'' are implemented.

The *virec* program basically extracts the ''undo'' version from the file. This is most useful when the system (or *elvis*) crashes in the middle of a long edit session, since the ''undo'' version of the file contains everything except your last change. Nearly all of your work can be salvaged.

The most common way to invoke *virec* is as follows. Give *virec* the name of the file you were editing, and it finds the matching file in */tmp* and writes the newest available version of the file over the existing version. It then deletes the */tmp* file.

The other way, where you redirect its standard input to come from a particular */tmp* file, is used when you have either forgotten which file that is and you want to see its contents, or when you want to recover the file without losing either the */tmp* file or the current version of the text file.

Options

−d tmpdir — If your *.exrc* or `EXINIT` string environment variable causes *elvis* to store temporary files in a non-standard directory (usually, any place other than */tmp*), then you must tell *virec* which directory to look in.

Files

*/tmp/elv** — The temporary file left over from when *elvis* was editing a text file.

See Also

elvis(U).

Name

wc − count characters, words, and lines in a file

Synopsis

```
wc [-clw] file ...
```

Description

Wc reads each argument and computes the number of characters, words and lines it contains. A word is delimited by white space (space, tab, or line feed). If no flags are present, all three counts are printed.

Options

−c Print character count.

−l Print line count.

−w Print word count.

Examples

```
wc file1 file2
```

Print all three counts for both files. This might look like:

```
  3       8     50 file1
254    2001  17344 file2
257    2009  17394 total
```

```
wc -l file
```

Prints the line count only. It might look like:

```
  3 file
```

Name

whereis − examine system directories for a given file

Synopsis

```
whereis file
```

Description

Whereis searches a fixed set of system directories, */bin*, */lib*, */usr/bin* and others, and prints all occurrences of the argument name in any of them.

Example

```
whereis stat.h
```

Should print: */public/module/ack/include/posix/sys/stat.h*.

Name

which − examine PATH to see which file will be executed

Synopsis

```
which name
```

Description

The PATH environment variable controls the search rules for executables. If a command *a.out* is given, the shell first tries to find an executable file in the working directory. If that fails, it looks in various directories, according to the value of PATH. The *which* command makes the same search, and gives the absolute path of the program that will be chosen, followed by other occurrences of the file name along the path.

Example

```
which a.out
```

Reports the path name of the *a.out* will be executed.

Name

who − report who is logged onto the system

Synopsis

```
who
```

Description

Who prints a list of the login names of the people logged in to Amoeba. It determines this by running *aps*(U) and looking for session servers. It extracts some relevant fields from the output of *aps* and prints them.

Warning

Who looks in the directory */public/hosts* to find out who is logged in. This is a very slow process if there are a lot of hosts which are down since it gets a short time-out while it tries to contact each dead hosts.

Example

```
who
```

might print

```
versto    armada00    session -d -a
keie      armada01    session -a
geels     armada01    session -a
phil      armada03    session -a
sater     armada03    session -a
leendert  armada03    -session -a /bin/sh
bahler    armada09    session -a
ast       armada09    session -a
kaashoek  vmesa4      session -a
gregor    vmesc2      session -a
```

See Also

whoami(U).

Name

whoami − Print username

Synopsis

```
whoami
```

Description

Whoami tells you in which protection domain it is run. This is done by comparing the `ROOT` capability environment variable with the capabilities in */public/users*.

Environment Variables

`ROOT` − the capability searched for, i.e., the capability for ‘‘/’’.

Files

/public/users − the directory with users home directories.

See Also

who(U).

Name

X Utilities − the set of X utilities that are known to work under Amoeba

Synopsis

```
The list of X utilities that work under Amoeba
```

Description

The following programs from X work under native Amoeba. For details about how they work and what they do, read the X manual.

atobm, bitmap, bmtoa, ico, maze, muncher, plaid, xcalc, xclock, xdpyinfo, xev, xfd, xkill, xload, xlogo, xlsfonts, xlswins, xmag, xmodmap, xprop, xrdb, xrefresh, xset, xsetroot, xwd, xwininfo, xwud

The following work with some slight reservations:

twm - has a problem with function keys on a vax, but this is very minor.

Name

xargs − execute a command with many arguments

Synopsis

```
xargs command [initial-arguments]
```

Description

Xargs is based on the UNIX System V command of the same name. This version performs only the default function of *xargs*, which is also the simplest function. It combines the *initial-arguments* with the arguments read from standard input and executes the *command* one or more times with those arguments.

Because UNIX imposes a limit on how much space the arguments of a command may use, if the argument list were too large the system prohibits the command from being executed (in System V, this error is `E2BIG`, *errno* = 7).

To avoid this problem, *xargs* will take arguments which are themselves a command and *its* arguments (options), and then read standard input and apply it to the command with its given arguments. *Xargs* is careful not to exceed the system-imposed limit, which is expected to be greater than the system's stream file buffer size, *BUFSIZ* in *stdio.h*. It continues to execute the command with the read-in arguments until it reaches end-of-file.

See Also

exec(L).

(UNIX ONLY)

Name

xlisten − X11 tool to listen for messages from user-to-user message service

Synopsis

```
xlisten [xterm-options]
```

Description

Xlisten is a shell script that invokes *listen* (U) in a separate *xterm* window. By default the window has geometry 80x12-0-0, i.e., it is a 12-line 80-column window in the bottom right corner of the screen. This can be overridden with a **−geometry** option. Any other options recognized by *xterm* are also recognized. In particular **−display**, but not **−e** which is used internally. Resources specified for *xterm* (through *xdm*) apply to xlisten.

To kill the client, type an interrupt in its window (typically ˆC).

For more information about the user-to-user message service, see *listen* (U), *tell* (U) and *topics* (U).

Warning

Your path must be set to find xterm and listen.

Example

To start the listen server in the background when you log in, put the following in your shell *.profile*.

```
xlisten &
```

See Also

listen(U), tell(U), topics(U).

Name

xreinit − Re-initialize an X server

Synopsis

```
xreinit display
```

Description

Xreinit causes the specified X server to re-initialize. Among other things, this causes all windows to be destroyed and all connections to be closed.

Warning

This call is specific to the Amoeba X server, so it will not work for X servers running on UNIX.

Example

The following line in your *.profile*

```
trap 'xreinit $DISPLAY' 0
```

will close all windows upon logout, leaving the X server ready for a new session.

See Also

X(U), xshutdown(U).

Name

xrlogin − Start a remote xterm

Synopsis

```
xrlogin [-ddd] hostname [-l username] [xterm-options]
```

Description

Xrlogin is a shell script that allows you to start xterm processes on remote machines. It uses *rush*(U) to connect to a remote machine via a TCP/IP connection, and executes *xterm*, the X terminal emulator. Since neither the *telnet* nor the *rlogin* protocols are supported on Amoeba (yet) *xrlogin* is the only method to get a shell on a UNIX machine from inside Amoeba.

Xrlogin attempts to leave as few processes around as possible, and is far superior to the naive `rsh host xterm &` in this respect.

The only required argument is the *hostname*. This is the host on which the *xterm* command should be started.

Options

−d 8 Normally, *xrlogin* will start everything in the background, thereby reducing the number of processes involved. This has the disadvantage that any (diagnostic) output may be lost. This option tells *xrlogin* to refrain from starting everything in the background so the diagnostics can be seen. More **−d** options give more debugging output, like commands executed, etc.

−l userSpecifies the remote user name that will be used to execute the *xterm* command. The default is taken from the USER environment variable.

Aside from these two all other options are passed to *xterm*.

Warnings

Since *xrlogin* uses *rush* you must have the machine where the TCP/IP server is running in your *.rhosts* file on the remote machine.

Xrlogin expects a Bourne-compatible shell on the remote machine. This means that *csh* users lose.

See Also

rush(U), xterm(X documentation).

Name

xshutdown − Perform orderly shutdown of an X server

Synopsis

```
xshutdown display
```

Description

Xshutdown performs an orderly shutdown of an X server. All connections are closed, and the server terminates. It is useful when shutting down a workstation.

Warning

This call is specific to the Amoeba X server, so it will not work for X servers running on UNIX.

Example

```
xshutdown myhost:0
```

terminates the X server on *myhost* cleanly.

See Also

X(U), xreinit(U).

Name

yacc − an LALR(1) parser generator

Synopsis

```
yacc [ -dltv ] [ -b prefix ] filename
```

Description

Yacc reads the grammar specification in the file *filename* and generates an LR(1) parser for it. The parsers consist of a set of LALR(1) parsing tables and a driver routine written in the C programming language. *Yacc* normally writes the parse tables and the driver routine to the file *y.tab.c*.

Options

The following options are available:

−b prefix This option changes the prefix prepended to the output file names to the string denoted by *prefix*. The default prefix is the character *y*.

−d This option causes the header file *y.tab.h* to be written.

−l If the **−l** option is not specified, *yacc* will insert #line directives in the generated code. The #line directives let the C compiler relate errors in the generated code to the user's original code. If the **−l** option is specified, *yacc* will not insert the #line directives. #line directives specified by the user will be retained.

−t This option will change the preprocessor directives generated by *yacc* so that debugging statements will be incorporated in the compiled code.

−v This option causes a human-readable description of the generated parser to be written to the file *y.output*.

If the environment variable `TMPDIR` is set, the string denoted by `TMPDIR` will be used as the name of the directory where the temporary files are created.

Tables

There is a program *:yyfix* that extracts tables from *yacc*-generated files. The program takes the names of the tables as its command-line arguments. The names of the tables generated by this version of *yacc* are *yylhs*, *yylen*, *yydefred*, *yydgoto*, *yysindex*, *yyrindex*, *yygindex*, *yytable*, and *yycheck*. Two additional tables, *yyname* and *yyrule,* are created if `YYDEBUG` is defined and nonzero.

Files

y.tab.c
y.tab.h
y.output
/tmp/yacc.aXXXXXX
/tmp/yacc.tXXXXXX
/tmp/yacc.uXXXXXX

Diagnostics

If there are rules that are never reduced, the number of such rules is reported on standard error. If there are any LALR(1) conflicts, the number of conflicts is reported on standard error.

Name

yap − yet another pager

Synopsis

```
yap [-num] [-cnqu] [+command] [filename ...]
```

Description

Yap is a program that allows the user to examine a continuous text one screen full at a time on a video display terminal. It does so by pausing after each screen full, waiting for the user to type a command. The commands are described below.

If no file name is given *yap* reads from standard input.

*Yap'*s main feature is that it can page both forwards and backwards, even when reading from standard input.

Options

−num An integer which is the size (in lines) of a page (the initial *page-size*).

−c Normally, *yap* will display each page by beginning at the top of the screen and erasing each line just before it displays on it. If your terminal cannot erase a line, *yap* will clear the screen before it displays a page.
This avoids scrolling the screen, making it easier to read while *yap* is writing. The **−c** option causes *yap* to scroll the screen instead of beginning at the top of the screen. This is also done if your terminal can neither erase a line or clear the screen.

−n Normally, *yap* recognizes escape sequences for stand-out mode or underlining mode in the input, and knows how much space these escape sequences will occupy on the screen, so that *yap* will not fold lines erroneously. The **−n** option suppresses this pattern matching.

−q This option will cause *yap* to exit only on the ‘‘quit’’ command.

−u Normally, *yap* handles underlining such as produced by nroff in a manner appropriate to the particular terminal: if the terminal can perform underlining well (ie., the escape sequences for underlining do not occupy space on the screen), *yap* will underline underlined information in the input. The **−u** option suppresses this underlining.

+command is taken to be an initial command to *yap*. (See the description of the commands below.)

Environment Variables

Yap uses the *termcap* database to determine the terminal capabilities and the default *page-size*. It examines the `TERM` environment variable to identify the terminal type. If `TERM` is not set, it defaults to *dumb*. *Yap* also examines the `TERMCAP` environment variable to locate the *termcap* database. If `TERMCAP` is not set, it defaults to */etc/termcap*.

Yap looks in the `YAP` environment variable to pre-set flags. For instance, if you prefer the **−c** mode of operation, just set the `YAP` environment variable to **−c**.

The commands of *yap* can be bound to sequences of key-strokes. The environment variable YAPKEYS may contain the bindings in the form of a list of colon-separated 'name=sequence' pairs. The *name* is a short mnemonic for the command, the *sequence* is the sequence of key-strokes to be typed to invoke the command. This sequence may contain a ^X escape, which means CTRL−X, and a \X escape, which means X. The latter can be used to get the characters '^', '\' and ':' in the sequence. There are two keymaps available, the default one and a user-defined one. You can switch between one and the other with the *change keymap* command.

Commands

The *yap* commands are described below. The mnemonics for the commands are given in parentheses. The default key sequences (if any) are given after the mnemonic. Every command takes an optional integer argument, which may be typed before the command. Some commands just ignore it. The integer argument is referred to as *i*. Usually, if *i* is not given, it defaults to 1.

visit previous file (*bf*) **P**
: Visit the *i*-th previous file given in the command line.

scroll one line up or go to line (*bl*) **^K** or
: If *i* is not given, scroll one line up. Otherwise, *i* will be interpreted as a line number. A page starting with the line indicated will then be displayed.

bottom (*bot*) **l** or
: Go to the last line of the input.

display previous page (*bp*) **−**
: Display the previous page, consisting of *i* lines, (or *page-size* lines if no argument is given).

display previous page and set pagesize (*bps*) **Z**
: Display the previous page, consisting of *i* lines, (or *page-size* lines if no argument is given). If *i* is given, the *page-size* is set to *i*.

scroll up (*bs*) **^B**
: Scroll up *i* lines (or *scroll-size* lines if *i* is not given. The initial *scroll-size* is 11).

search backwards for pattern (*bse*) **?**
: Search backwards for the *i*-th occurrence of a regular expression which will be prompted for. If there are less than *i* occurrences of the expression, the position in the file remains unchanged. Otherwise, a page is displayed, starting two lines before the place where the expression was found. The user's erase and kill characters may be used to edit the expression. Erasing back past the first character cancels the search command.

skip lines backwards (*bsl*) **S**
: Skip *i* lines backwards and display a page.

skip pages backwards (*bsp*) **F**
: Skip *i* pages backwards and display a page.

scroll up and set scrollsize (*bss*) **b**
: Scroll up *i* lines (or *scroll-size* lines if *i* is not given. If *i* is given, the *scroll-size* is set to *i*.

change key map (*chm*) **^[**
: Change from the current key map to the other (if there is one).

exchange current page and mark (*exg*) **x**
: Set the mark to the current page, and display the previously marked page.

visit next file (*ff*) **N**
: Visit the *i*-th next file given in the command line.

scroll one line down or go to line (*fl*) **^J** or
: If *i* is not given, scroll one line down. Otherwise, *i* will be interpreted as a line number. A page starting with the line indicated will then be displayed.

display next page (*fp*) **<space>**
: Display the next page, consisting of *i* lines, (or *page-size* lines if no argument is given).

display next page and set pagesize (*fps*) **z**
: Display the next page, consisting of *i* lines, (or *page-size* lines if no argument is given). If *i* is given, the *page-size* is set to *i*.

scroll down (*fs*) **^D**
: Scroll down *i* lines (or *scroll-size* lines if no argument is given).

search forwards for pattern (*fse*) **/**
: Search forwards for the *i*-th occurrence of a regular expression which will be prompted for. If there are less than *i* occurrences of the expression, the position in the file remains unchanged. Otherwise, a page is displayed, starting two lines before the place where the expression was found. The user's erase and kill characters may be used to edit the expression. Erasing back past the first character cancels the search command.
: Note: Some systems do not have *regexp*(L). On those systems, searches are still supported, but regular expressions are not.

skip lines forwards (*fsl*) **s**
: Skip *i* lines and display a page.

skip pages forwards (*fsp*) **f**
: Skip *i* pages and display a page.

scroll down and set scrollsize (*fss*) **d**
: Scroll down *i* lines (or *scroll-size* lines if *i* is not given. If *i* is given, the *scroll-size* is set to *i*.

help (*hlp*) **h**
: Give a short description of all commands that are bound to a key sequence.

set a mark (*mar*) **m**
: Set a mark on the current page.

repeat last search (*nse*) **n**
: Search for the *i*-th occurrence of the last regular expression entered, in the direction of the last search.

repeat last search in other direction (*nsr*) **r**
: Search for the *i*-th occurrence of the last regular expression entered, but in the other direction.

quit (*qui*) **Q** or
: Exit from *yap*.

redraw (*red*) **^L**
: Redraw the current page.

repeat (*rep*) **.**
: Repeat the last command. This does not always make sense, so not all commands can be repeated.

shell escape (*shl*) **!**
: Invoke the shell with a command that will be prompted for. In the command, the characters '%' and '!' are replaced with the current file name and the previous shell command respectively. The sequences '\%' and '\!' are replaced by '%' and '!' respectively. The user's erase and kill characters can be used to edit the command. Erasing back past the first character cancels the command.

pipe to shell command (*pip*) **|**
: Pipe the current input file into a shell command that will be prompted for. The comments given in the description of the shell escape command apply here too.

go to mark (*tom*) **'**
: Display the marked page.

top (*top*) **^^**
: Display a page starting with the first line of the input.

visit file (*vis*) **e**
: Visit a new file. The file name will be prompted for. If you just type a return, the current file is revisited.

write input to a file (*wrf*) **w**
: Write the input to a file, whose name will be prompted for.

The commands take effect immediately, ie. it is not necessary to type a carriage return. Up to the time when the command sequence itself is given, the user may give an interrupt to cancel the command being formed.

Diagnostics

One hopes these are self-explanatory.

Files

/etc/termcap − terminal capabilities data base.

Warnings

Yap will find your terminal very stupid and act like it, if it has no way of placing the cursor on the home position, or cannot either erase a line or insert one.

In lines longer than about 2000 characters, a linefeed is silently inserted.

The percentage of the file read, given in the prompt when *yap* reads from a file (and knows it), is not always very accurate.

Example

```
yap myfile
```

will display the text in *myfile* on the screen.

See Also

termcap(L), regexp(L).

8 Index

This will be a permuted index derived using a cross-reference generator which will do things by keywords and phrases and/or apropos strings or something sensible.