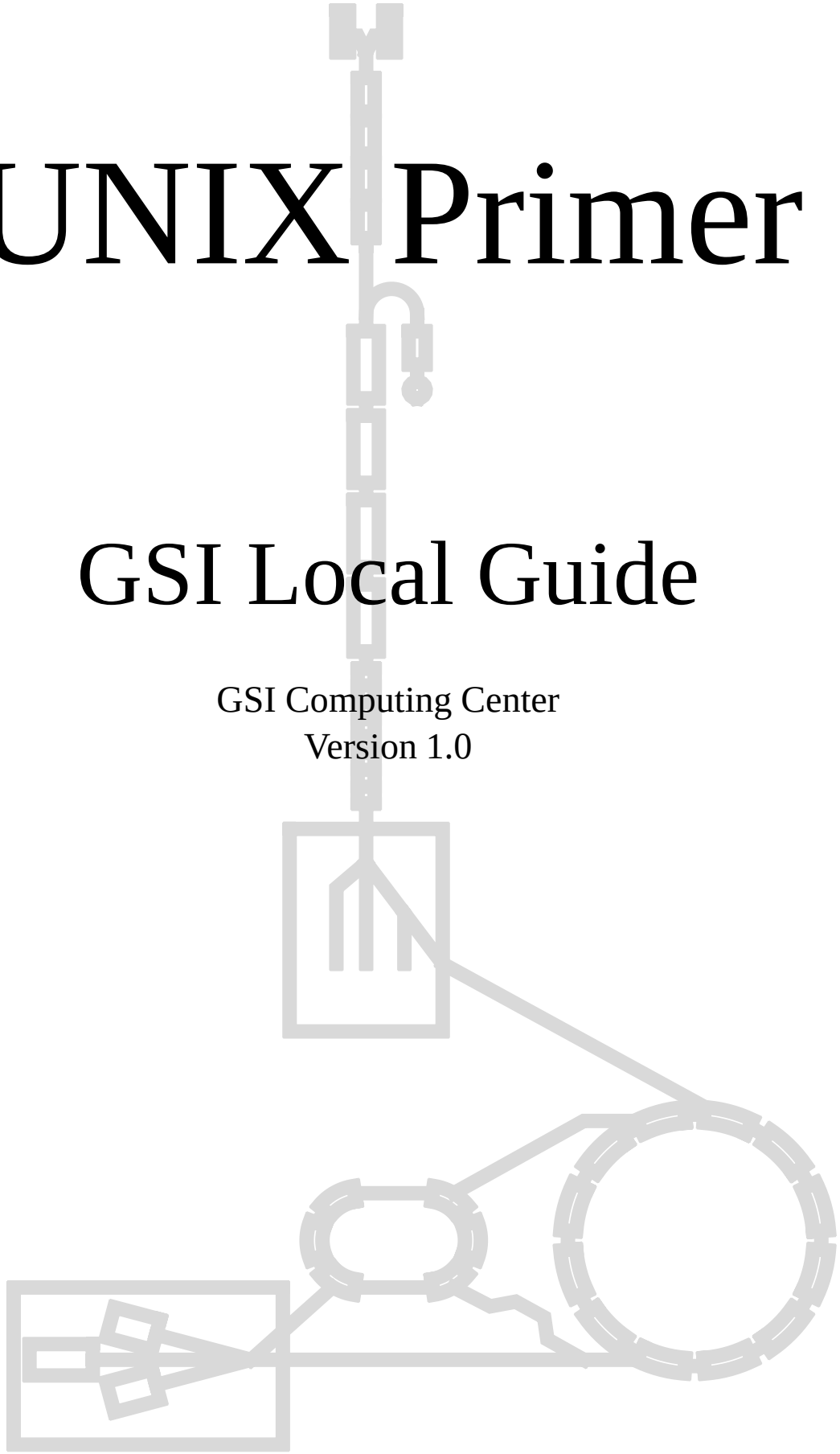
UNIX Primer

GSI Local Guide

GSI Computing Center

Version 1.0



Preface:

In early summer 1991 the GSI Computing Center started a Unix Pilot Project investigating the hardware and software possibilities of centrally operated unix workstation systems. A few machines from DEC, HP and IBM have been installed in the central computer room and have since been used as development platforms by the computer centers application software-, telecommunication-, and systems groups. We feel that now the moment has come to publish our experience with the pilot project. At the same time we want to make the machines available to a larger number of users at GSI and - hence - have chosen the form of this booklet, the

Unix Primer - GSI Local Guide

as our publication medium. We do not consider the present version as complete but rather as a working document, which will be updated at regular intervals. As you can easily see when reading, the primer is the product of many authors. A common style is only evolving. However, as it gives the answer to many "frequently asked questions" we have chosen to publish version 1.0 in the state it is in now. We hope that nevertheless the **Primer** will be helpful to many users. Any readers are welcome to comment and make suggestions. We will gratefully accept mail, printed copies with corrections or personal visits at the help desk (alias Benutzerberatung).

The following people have made contributions to version 1.0 of the primer: Michael Dahlinger, Hans Döbbeling, Horst Göringer, Richard Herrmann, Eva Hocks, Frank Kraske, Peter Malzacher, Udo Meyer, Thomas Schwab, Heiko Weber.

This document has been produced with the LaTeX macro package. The style used (gsiman.sty) has been derived from CERN's cernman style. We acknowledge the support of M.Goossens and A.Samarin [4] from CERN/CN.
Michael Dahlinger has coordinated this project and has taken care of the final editing.

Printed copies of this document can be obtained from the computer centers help desk. Print your own copy by

```
lp -d pshpad /usr/local/doc/primer/primer.ps  on HP-UX or
lpr -P pshpad /usr/local/doc/primer/primer.ps  on AIX
```

A copy of this document can also be obtained via anonymous ftp on internet node **ftp.gsi.de** as `/dist/doc/primer.ps`.

Unix Primer, Version 1.01 printed at GSI: 7th December 1992

If You Need Help

There is a help desk **(Benutzerberatung)** for all general computing related questions.

If you need help, please contact the help desk first.

Telephone 555, Office 2.244, Südbau, first floor
The opening hours are 9:30 to 11:30 and 14:00 to 17:00 except for Tuesdays (from 10:30) and Fridays (until 16:00). If the help desk cannot help you, they will refer you to the appropriate specialist.

If you have Hardware Problems:

For hardware and networking problems, repairs, printer maintenance, please contact the central operating Tel. 515, machine-room, Südbau, ground-floor, Monday – Friday 6:00 to 23:00.

Service	Name	Office	Phone	E-mail
General Help Desk	Benutzerberatung	2.244	555	
User Registration	Eva Hocks	2.248	519	hocks@rzri6f.gsi.de
X-Terminal Installations	Operations	1.250	515	
Name Service	Udo Meyer	1.249	525	rz02@mvs.gsi.de
System Management Unix	Eva Hocks	2.248	519	hocks@rzri6f.gsi.de
System Management AIX	Heiko Weber	2.247	556	weber@rzri6f.gsi.de
System Management HP-UX	Horst Göringer	2.245	553	goeri@rzhp9a.gsi.de
System Management ULTRIX	Hans Döbbeling	2.246	554	dob@rzhp9a.gsi.de
CERN Software Support	Michael Dahlinger	2.235	546	dahlinge@rzhp9b.gsi.de
Public Domain Software, GNU	Thomas Schwab	2.245	553	schwab@rzri6b.gsi.de
Software Support	Peter Malzacher	2.223a	551	peter@rzri6b.gsi.de
Software Support	Axel Möller	2.223a	551	moeller@rzri6b.gsi.de
LaTeX on Unix	Michael Dahlinger	2.235	546	dahlinge@rzhp9b.gsi.de
X-Terminal General Info	Hans Döbbeling	2.246	554	dob@rzhp9a.gsi.de
Networking Questions	Udo Meyer	1.249	525	rz02@mvs.gsi.de
Networking, NFS & FTP	Frank Kraske	1.262	514	kraske@rzri6f.gsi.de
Graphics Tools on Unix	Richard Herrmann	2.237	548	rz45@mvs.gsi.de

Manual Conventions:

Throughout this manual the following typographical conventions are used:

If You See This ...	It Means ...
$ `what you type`	`underlined` text is used in dialog examples to distinguish what you type from what the computer displays
$ `command` *parameter*	*italic* text is used to denote parameters which must be specified on commands
`<ESC>-q`	you press the **escape key** on the keyboard followed by key **q**. The escape key usually sits on the top left of a PC-like keyboard or on the top center coinciding with **F11** on a VT200-like keyboard
`<Ctrl-c>`	you press and hold-down the **control key** and simultaneously press the **c** key.
`<Alt-d>`	you press and hold-down the **alt** key and simultaneously press the **d** key. The **alt** key - if it exists - is next to the space bar on both VT200-like and PC-like keyboards
`<Tab>`	you press the **Tab** key
`<Backspace>`	you press the **Backspace** key
`<Del>`	you press the **Delete** key
Mb1	the left mouse button
Mb2	the center mouse button
Mb3	the right mouse button

Table of Contents

List of Figures

List of Tables

Chapter 1: How to get started

Currently the GSI computing center supports Unix platforms RS/6000 (IBM), HP 9000-700 (Hewlett Packard), and DEC 5000 (Digital Equipment) with the operating systems AIX, HP-UX, and ULTRIX, respectively. Users can get in contact with Unix on three different levels:

- The simplest access is when starting a Unix session with the `telnet` command from a **text oriented** terminal, such as a DEC VT100 or VT200, for example.
- With the appropriate hardware platform - an X-terminal, PC, or workstation - users can work in a comfortable **X-Windows** environment. The X-Windows system is a network-based graphics windowing system, which can be seen as a layer above the operating system.
- You can work in an even more comfortable (X-Windows) environment, if you use a **session manager**, which controls your complete session with all applications running in it. So it is very easy, for example, to restore the previous session or to lock the current session, and to organize your working environment in several workspaces. Session managers are available with the Visual User Environment (VUE) of HP, or with dxsession of DEC.

1.1 How to obtain an Account

If you want to work with Unix, please contact one of the corresponding system managers to get an account:

AIX Eva Hocks, room 2.248, tel 519, hocks@rzri6f.gsi.de

Heiko Weber, room 2.247, tel 556, weber@rzri6f.gsi.de

HP-UX Eva Hocks, room 2.248, tel 519, hocks@rzri6f.gsi.de

Horst Göringer, room 2.245, tel 553, goeri@rzhp9a.gsi.de

ULTRIX Hans Döbbeling, room 2.246, tel 554, dob@rzhp9a.gsi.de

Currently the following Unix hosts are offered by the computing center:

rzhp9a HP 9000-720
rzhp9b HP 9000-720
rzri6a IBM RS/6000-320
rzri6f IBM RS/6000-970
rzds5a DEC 5000-200

For more details on the currently available hardware see C.1 on page 64.

1.2 Login

The steps to be performed in the login procedure depend on your working place. Vendor dependencies are negligible. However, there are some differences when logging in on 'dumb' text oriented terminals, or on devices supporting X-windows such as workstations, PCs, or X-terminals, especially when working in environments under control of a session manager.

1.2.1 Text oriented Access to Unix

Login via Telnet

If you want to access a Unix workstation, e.g. rzhp9a running HP-UX, enter the command

```
telnet rzhp9a
```

There appear some messages and finally the login prompt:

```
Trying...148.181.64.51
Connected to RZHP9A
Escape character is '^]'.

HP-UX rzhp9a A.08.07 E 9000/720 (ttys7)

login:
```

Enter your account name, and the password will be prompted:

```
password:
```

Be careful when entering your account name and password, because Unix is **case sensitive**! Upper and lower case letters have a different meaning!

If you have an valid account on rzhp9a, the system greets you with a welcome message and provides some information, such as the the existence of unread news or new mail files. Then you are asked for two things:

1. your **terminal type** (see next section), and

2. your **display address**. Because your terminal has no display address, just press <Enter> at this prompt. For more details, see section 1.2.2 on page 3.

Now the system responds with the default prompt

hostname:/u/*username sequencenumber*$

or, to give an example, with

```
rzhp9a:/u/goeri 33$
```

and is ready for command input.

Terminal Type

With the terminal type you describe the terminal hardware or the emulation program you use when communicating with Unix. The most important terminal types used at GSI are

vt100	DEC
vt220	DEC
3270	IBM [1]
xterm	standard X-window terminal
aixterm	AIX terminal emulation
hpterm	HP-UX terminal emulation
dxterm	ULTRIX terminal emulation

If you work in a Unix environment with the wrong terminal type set, you should keep in mind that not all keys of your keyboard may be available in the way you expect it! Then you should correct your terminal type with the `export` command (see section 1.7.1 on page 8). For example, if you want to correct your terminal type to vt100, you have to enter

```
export TERM=vt100
```

On dumb terminals, there may be applications that can not work correctly, because they require hardware features not available. However, many Unix commands can be used in such an environment, e.g. to compile and run user programs, or to look into the file system.

Logging in from MVS

Access to a Unix workstation via telnet is also possible from a 3270 terminal under MVS. However, the login procedure is a little bit more complicated, and you can only work in line mode. As the invocation of full screen applications (e.g. the editors vi or emacs) may hang up your session, this access is not recommended in general. If your session hangs up, hit one of the buttons <F4> to <F12>. Then the prompt

```
Telnet command:
```

appears, and you can finish the current telnet session with the `close` command.

For example, to access from your 3270 terminal the RS/6000 workstation rzri6f, enter the TSO-command

```
telnet rzri6f
```

Now TSO confirms the action:

```
TCPTEL001I MVS TCP/IP Telnet V2R1
TCPUTM110I Connecting to RZRI6F
            140.181.64.35, port TELNET (23)
***
```

Press <Enter>, and the following information appears:

```
TCPUTM118I
TCPUTM119I Using Line Mode...

TCPUTM120I
TCPUTM121I Notes on using Telnet
     when in Line Mode:
TCPUTM122I - To hide Password,
     Hit PF3 or PF15
TCPUTM123I - To enter Telnet Command,
     Hit PF4-12, or PF16-24
***
```

After pressing <Enter>, a new screen appears. The top line is

```
AIX telnet (rzri6f)
```

The bottom line is

```
Telnet command:
```

From now, the bottom line is the command line for the duration of your Telnet session. At this time, however, you must **not** enter a Telnet command, but just press <Enter>. The top line remains unchanged, and the login prompt appears:

```
login:
```

Enter your account name at the current cursor position, and the password prompt appears.

```
Password:
```

To hide your password, enter at first <F3>, as indicated in the information screen above, and then type your password. Be careful when entering your account name and password, because Unix is **case sensitive**! Upper and lower case letters have a different meaning!

[1] only known under AIX

If you have a valid account on rzri6f, the system greets you with a welcome message and provides some information, such as the the existence of unread news or new mail files. When prompted for your terminal type, enter `3270`. Because your terminal has no display address, just press `<Enter>` when asked for your display address. Then the system responds with the corresponding default prompt as in the example above and is ready for command input.

After some command inputs, the bottom line of your screen shows the string **Holding** on the right, and further inputs are not possible until you press the **PA1 key**.

1.2.2 Accessing Unix via X-Windows

You can skip this section, if you work with Unix via dumb terminals such as DEC VTxxx or IBM 3270. This section is only relevant for hardware platforms enabling the usage of X-windows, such as X-terminals, PCs, or workstations.

X-Windows is a standardized, vendor independent user interface, originally developed at MIT. It is a network-based graphics windowing system, which allows you to work with multiple programs simultaneously, each in a separate window of your screen. Especially you may have access to different hosts from different windows, but on a single physical screen. This includes also the non-Unix operating systems available at GSI such as MVS on IBM 3090 (see 4.4 on page 23) and VMS on the VAX computers.

X-server

Central part of the X-Window system is the X-server, also called X-Window server or display server. It controls screen, keyboard, and mouse, and the process communication requests. The X-server updates the windows on the screen on request of the clients (e.g. programs initiated by your input).

On X-Terminals, the X-server is already locally available and need not be started once more. If you work locally at a workstation or a PC, you may have to start the X-server by yourself. The command name depends on the X-emulation used. On a RS/6000 workstation, for example, this can be done with the command

```
xinit
```

Terminal Windows

At the command prompt after login, you can start windows with **terminal emulations** (terminal windows) using the command

```
xt
```

which is available on all Unix hosts owned by the GSI computing center. Depending on the hardware platform, **aixterm, hpterm**, or **dxterm** commands, respectively, are invoked with GSI-specific defaults. They are labeled on top with

hostname:username

X-terminal users should **not** work within telnet windows running VT100 emulations (see section 'Access via X-Terminal' below), but only within terminal windows running adequate terminal emulations. In telnet windows, only a subset of keys and fonts is available, and several tools (e.g. when requiring the use of function keys) may not work correctly. **The intended use of telnet windows is only login!**

Working with multiple windows requires a window manager, which allows, for example, to change the size and position of windows. The window manager provided with **OSF/Motif** is called **mwm**, the Motif window manager. Its functionality is described in 1.6.2 on page 7.

Display Address

The display address is the address of the 'host', on which your X-server resides, that means, the address of your X-terminal, PC, or workstation, appended with `:0`. The host address has been assigned during installation and can be the **internet address or name**.

Examples of display addresses at GSI are:

xwtae:0	(X-terminal)
pc01:0	(PC)
vsaa:0	(VMS Vaxstation)
dsaa:0	(ULTRIX workstation)
140.181.97.168:0	(decimal representation of display address)

If you have a Unix working place for your own, you should enter the display address in your profile file. This will be discussed in more detail in section 1.7 on page 8.

Access via X-Terminal

If your X-terminal is up and running, use a free **telnet window** for login. In a telnet window, in principle a VT100 terminal emulation program is active. It is labeled on top with **telnet**, and if it prompts you with

```
telnet>
```

you can use it to login. In case of another prompt the window is already in use by an active login session. If there is no free telnet session available, you must create a new telnet window.

On the Tektronix X-terminals at GSI, this can be done via the following procedure:

1. Press the **Setup** key on your keyboard (first row, rightmost key) - the setup menu will appear.
2. If there is a frame around the menu, switch to item 4. If **not**, move the mouse cursor to the field **Local Clients** (top bar) and press the left mouse button. The Local Clients menu appears.
3. Keep the button pressed, move the mouse cursor within this menu to the field **MWM** to activate the local Motif window manager and then release the mouse button again: after a few seconds, the setup menu (and all following windows in your session) will have an Motif frame (see section 1.6.2 on page 7.
4. Move the mouse cursor to the field **Sessions** (top bar) and press the left mouse button. The Sessions menu appears.
5. Keep the button pressed, move the mouse cursor within this menu to the field **Telnet** and then release the mouse button: a telnet window with the appropriate prompt `telnet>` will appear.

Now you can select the logical name of the host and login using the **open** statement:

`telnet>` `open` *`hostname`*

If specifying a valid host name (e.g.rzri6f), the connection will be established, and the login prompt appears:

```
IBM AIX Version 3 for RISC System/6000
(C) Copyrights by IBM and by others 1982, 1991

login:
```

To initiate the login process, at first you have to authorize yourself by entering your account name and password. Layout of the login panel and the system prompts may vary slightly depending on the connected host. In all cases, however, your password, is not echoed when typing.

Be careful when entering your account name and password, because Unix is **case sensitive**! Upper and lower case letters have a different meaning! Besides that, you should keep in mind that the normal backspace keys might not yet be available during the login process in some environments! In this case, reenter name or password completely.

If you enter your account name and password correctly, the login process proceeds. You receive a welcome message and some information from your host, such as the existence of unread news or of new mail files.

Afterwards you are asked for the terminal type. (see section 1.2.1 on page 2). Depending on the selected host, the appropriate defaults are offered:

aixterm on AIX
hpterm on HP-UX
dxterm on ULTRIX

Just press `<Enter>` to get the appropriate value.

Then you have to specify the display address of your terminal (see section 1.2.2 on page 3).

Now the system responds with the default prompt

hostname`:/u/`*username sequencenumber*`$`

or, to give an example, with

```
rzri6f:/u/goeri 33$
```

and is ready for command input. The default prompt can be changed individually in your profile file (see section 1.7 on page 8).

1.2.3 Accessing Unix in an HP-VUE Environment

If you have an X-terminal, and if it is up and running, a dedicated HP workstation is already connected to your X-Terminal. A login menu with fields for your account name (username) and password is offered.

Be careful when entering your account name and password, because Unix is **case sensitive**! Upper and lower case letters have a different meaning!

If you enter your account name and password correctly, the login process proceeds. You receive a welcome message and some information from your host, such as the existence of unread news or of new mail files.

If you log in with HP-VUE for the first time, a lot of menus and icons appears for immediate graphics access with mouse and click. For more information, see the section 1.4 on page 5.

At a following login you get a default startup environment, or you can continue with the previous environment which was active at the last logout - depending on what you said when logging out.

1.3 Unix Shells

If the login process has successfully completed, you are in an environment called **shell**. It is another process, which has been started (**spawned**) at the end of the login process. A new shell is also started for each invocation of a terminal window.

A shell is the interface between the operating system and the user. It interprets the commands you type, and the keys you press, in order to direct the operating system to take an appropriate action.

On most platforms, there are several shells available differing in power and functionality. The most important ones are

- the **Bourne Shell** (sh),
- the **Bourne Again Shell** (bash),
- the **C Shell** (csh), and the
- the **Korn Shell** (ksh).

Most Unix systems offer several shells with the Bourne shell as default. However, at the Unix platforms supported by the GSI computing center, you have the Korn shell as default, and all GSI customization is only done for the Korn shell (see section 1.7 on page 8). It is a newer shell, developed by David Korn at Bell Laboratories, and it is upwardly compatible with the most features of the Bourne shell.

You can invoke a new Korn shell (subshell) on top of your current shell by typing

```
/bin/ksh
```

e.g., to get a new Korn shell. You can exit this subshell again by typing

```
exit
```

If you repeat the `exit` command once more, your terminal emulation is closed, and the terminal window disappears. Instead of `exit`, you can also enter `<Ctrl-D>`.

More details on the Korn shell environment and its customization can be found in 1.7 on page 8.

1.4 Desktop Environments

A desktop environment provides a collection of windows with menus and icons for immediate graphical access (with mouse and click) to the most useful commands and applications. If you click a menu button or icon, application programs are started providing the required functionality. Depending on the task they fulfill they may have names such as file manager, style manager, help manager, and so on.

A desktop environment may be invoked by the user via command in an existing session, such as AIXwindows Desktop, or it may even control the complete session, such as HP VUE, via its session manager. Therefore, if HP VUE is enabled on your workstation or X-terminal, you can login via a special login panel and finally logout again by activating a logout field. The default configuration of such environments can be tailored by each user for his specific needs.

Each desktop contains a collection of small pictures, called **icons**, which visually contain the contents of the desktop. Each icon visually represents an actual file, directory, program, or object and can be activated by double click with the left mouse button. If you activate a program icon, you actually run the program. If you (double) click an icon representing a directory, the contents of this directory is presented in a new window, either with text lines or with new icons.

Desktops also contain **menu bars** with menu fields, which can be activated by single click with the left mouse button. In general they can be used to create new files, directories, or other desktops, and help information.

The following three paragraphs give a very brief overview on the desktop environments available on the three Unix flavors supported by the GSI computing center. For more information on the functionality and on the configuration possibilities activate the **Help** menu in the specific desktop environment.

1.4.1 AIXwindows Desktop

AIXwindows Desktop runs on RS/6000 workstations under AIX. Two different desktops are offered each preconfigured to meet a particular skill level:

1. The **General desktop** meets the needs of users with little or no AIX background.
2. The **Power desktop** provides ready access to AIX commands and the AIX file system for users with AIX skills.

AIXwindows Desktop can be invoked with the command

```
xdt3
```

After a greeting menu, the Power desktop appears as default.

1.4.2 HP Visual User Environment

HP VUE runs on Hewlett Packard 9000-700 workstations with HP-UX and X-Windows. It is a set of enhancements to the X-Windows system which controls complete sessions from login to logout. In addition to AIXwindows Desktop, it offers also components such as a

1. **Login manager:** Performs configuration activities, starts the X-server, and performs the login and logout activities.

2. **Workspace manager:** Controls and manages up to six different working environments ('workspaces'), which can be set up by the user.

You can lock your keyboard for times of absence by activating an icon. To unlock again, you must enter your password. The complete configuration of your session can be saved when logging out and restored again after a new login (see section 1.5 on page 6.)

1.4.3 Dxsession

Dxsession will be described in a later release of this document.

1.5 Logout

Closing a Terminal Window

When you are finished with your work session, you can close the shell(s) running in your terminal window(s) by

- entering the shell command **exit**
- entering `<Ctrl-D>`, or
- using the **Close** button of the **window menu**. The window menu is activated by the button in the upper left corner. Move the mouse cursor there and press the left mouse button. The window menu appears. Now move the cursor to the field **Close** and press the left mouse button again. The handling of the window menu is described in more detail in section B on page 62.

A terminal window disappears when the shell running in this window is closed. A session is ended when the primary login shell is closed.

Closing a Telnet Window

If the login shell in a telnet window is closed, the connection to the host is broken. The window is still available and prompts you with `telnet>` - ready for the next login.
You can also close a telnet window (without host connection) by

- entering **quit**,
- entering `<Ctrl-D>`, or
- utilizing the **Close** button of the **window menu**.

The telnet window also disappears then.

Closing a HP-VUE Session

If you work with HP VUE, it is not necessary to close any window for logout. You just need to press the logout field, the rightmost button in the functions menu at the bottom of your screen. With the **Style Manager**, you can specify what happens each time you press the logout field:

- Either the current window configuration will be stored and be available again after the next login, or
- you will start with a default window configuration after the next login, or
- you are asked, if you will start next time with the actual or default window configuration.

1.6 Some initial Hints

1.6.1 The first Commands

To run a command, type the command's name after the prompt and press the `<Enter>` key. When the command is finished, the prompt appears again. For example, if you want to know who and where you are:

```
rzhp9a:/u/goeri 34$ whoami <Enter>
goeri
rzhp9a:/u/goeri 35$ hostname <Enter>
rzhp9a
rzhp9a:/u/goeri 36$
```

You get the answers, and the command number is incremented with each new prompt.
Another important command is `cd`, which stands for change current working directory:

```
rzhp9a:/u/goeri 35$ cd util <Enter>
rzhp9a:/u/goeri/util 36$
```

Shell Scripts

The commands described in the previous examples have been executed when typed at the command prompt. They can also be written to files and executed as command lists or **shell scripts**, which corresponds to a TSO CLIST in MVS or a DCL in VMS. For more information, see section 2.4 on page 13.

Set Password

Creating your account on a host, the system administrator gives you a start-up password. Because it is easy to guess, you should change it immediately when you have logged in the first time using the command

```
passwd
```

After invocation, you are at first prompted for the current password to check your authorization. Then you can enter a new password. You have to type it in twice, because the system compares both versions to avoid accidental mistyping. From security reasons, the new password has to fulfill some requirements:

- It must have at least six characters. If you enter more than eight characters, only the first eight characters are significant.
- Each password must contain at least two letters (upper or lower case) and at least one number or special character.

  Please keep in mind that **upper case and lower case letters are treated as different**.
- A password must be different from the account name and any reverse or circular shift of it.
- A new password must be different from the old one and any reverse or circular shift of it.

If you have forgotten your old password, please contact the system administrator of the corresponding host (see 1.1 on page 1).

1.6.2 Handling Motif-Windows

In each window, a process or program is active. In a terminal window, a **terminal emulation** running a shell is active (see 1.2.2 on page 3). Other windows can be created by application programs to communicate with the user.

You can have multiple windows on your screen, which can **overlap** much like sheets of paper on a desktop. Both, the terminal and application windows, have a frame in **Motif** style and are managed by the Motif window manager (mwm) provided with OSF/Motif.

Windows are handled by **pointer devices**, that means input devices, that, when moved across a flat surface, move the pointer symbol on the display correspondingly. Pointers usually have buttons that can be pressed to send signals, which in turn accomplish certain functions. The **mouse** is the most common example of a pointer device.

With a mouse, a lot of functions can be performed with windows:

1. You can accomplish **geometrical** operations, such as moving or resizing.
2. The contents of windows can be **scrolled**.
3. Windows can be **iconized**, that means, they can be converted to a small picture called **icon**, which is a representation of an inactive window, and they can be restored again into the state of an active window.

In section B on page 62 the operations possible with Motif windows are explained in detail.

Using the Mouse Cursor

The mouse is used to indicate or activate a graphical element on the screen such as a window, icon, or command button. Normally, a mouse has three buttons labeled here as left, middle, and right button. By placing the mouse on a particular element and then performing some button action and possibly cursor motion, you can invoke a variety of commands. The types of actions you can perform are:

Click Press the button down and release it immediately again. A double or triple click is two or three clicks in succession, with no pause between clicks.

Press Push the button and hold it down.

Release After pressing a button down, release it by letting up on the button.

Drag To drag a graphical object (e.g. a window or an icon) from one location on the screen to another, place the cursor on the appropriate field of the object. Then press the mouse button and move the cursor to the required new location, such dragging the object. Then release the button again.

Motif Widgets and Menu Types

This short paragraph provides some brief definitions of some terms which are often heard in connection with X-Windows. Because the exact knowledge of these terms is not required when using X-Windows, this paragraph can be skipped by newcomers.

OSF/Motif provides a set of user interface mechanisms that are composed of data structures and procedures. They can be displayed in many ways, for example as buttons, boxes, labels, scroll bars, and so on, and are called **Motif widgets**.

However, in public domain software, often another class of widgets, called **Athena widgets**, is used. Their implementation and graphical visualization differs somewhat from the Motif widgets.

The Motif widget set provides three types of menus:

1. **Pulldown Menus**: are displayed as horizontal menu bars with (normally) several labels. If you activate a label, it will display a vertical

2. **Popup Menu**. A popup menu also offers several labels to select. It disappears again when released after execution.

3. **Option Menus** are very similar to popup menus and offer a one-of-many selection. However, when the selection is complete, the selected label remains visible in the menu, and only the not selected alternatives disappear.

Active Windows

The window which currently accepts input is called the **active** window and differs from all other windows by a colored frame. The active window sits on top of the window stack. The other windows, sometimes referred to as **background** windows, currently do not accept input. The processes running in the background windows, however, are still active (in background!) and are even able to produce output. To direct input to another window, just move the mouse to any part of it and click with the left mouse button.

Standard GSI Terminal Windows

A standard GSI terminal window can be obtained with the command

```
xt
```

It is an Motif window and has an outer **frame** and a horizontal **title bar** on top and a vertical **scroll bar** at the right, but both within this frame. They consist of several parts with each of them enabling specific functions. The mouse cursor, which has the shape of an 'I' when positioned within the text field, changes its appearance when moved to one of these elements. The new shape depends on the specific location.

1.7 GSI Customization

1.7.1 Environment Variables

There are parameters in the shell that define parts of your working environment and can be set interactively at the command prompt, in shell scripts, or in user specific profile files.

- Shell parameters that are **local** to your login shell and not passed to any subshell or subprocess are called **shell variables**.

- Shell parameters that are **global** are called **environment variables**. They are valid in the current shell, where they are set, and in all subshells. They are not valid, however, in 'higher' shells, from where the current shell is invoked as subshell. To be global, shell parameters must be **exported** (see below).

There is one restriction, if you define environment variables in a shell script (see 2.4 on page 13). You must invoke the script with a preceding dot, if the environment variables shall be valid also in the current shell:

```
. myscript
```

Else the environment variables are valid only within your script and all subprocesses invoked from there.

A set of environment variables is already defined by the operating system and can be changed by each user for his personal working environment. In addition, you can also define new environment variables, of course. The profile files as an interface for initialization at login are described below.

A list of the currently valid environment variables may be obtained with the command

```
printenv
```

In the following examples, for the reasons of simplicity and shortness,the command prompt is assumed to be `$` .

Setting Environment Variables

Your default printer, for example, is defined by the environment variable **LPDEST** (see section 4.5 on page 23 for information on the available printers and their names and characteristics). To change its default value `pshpad` to `pshpa`, enter the command

```
$ export LPDEST=pshpa
```

Getting Values of Environment Variables

The contents of environment variables can be made visible with the `echo` command. To get the value, the name of the environment variable must be preceded by a $-sign:

```
$ echo $LPDEST
pshpa
```

The value `pshpa` is shown (assuming that you didn't change the system default value).
If you forget the $-sign, not the value, but the character string is printed:

```
$ echo LPDEST
LPDEST
```

Again you should keep in mind that Unix is case sensitive, that means environment variables such as `lpdest` or `LPdest`, for example, are not defined normally.

A More advanced Example

For example, if you want to have your current host name and working directory available in an environment variable named, `MyEnv`, you can proceed as follows:

```
$ export NODE=‘hostname‘
$ export MyEnv=$NODE:$PWD
$ echo 'My environment: $MyEnv'
My environment: rzri6f:/u/goeri
```

With the first statement, a new global variable `NODE` is defined. `NODE` gets the output of the command `hostname` as value, which is achieved by enclosing the command name in single backquotes. Then the values of `NODE` and of the environment variable `PWD`, which is already provided by the system and contains always the current working directory, are put together in `MyEnv`. Finally, to control the success of this action, the contents of `MyEnv` is printed in your terminal window, prefixed by some text.

A Summary of some useful Environment Variables

In the following, some useful environment variables provided by the system are listed in alphabetical order:

EDITOR: The default editor is **vi**.

ENV: The name of a shell script that is executed each time when a new shell is invoked. This shell script is used, for example, to define common alias names, which should be available through the whole environment. A common default in the Korn shell environments is

```
~/.kshrc
```

The tilde '~' specifies your home directory. This file is also in use at GSI for your private shell customization. However, in order to provide a mechanism for common and platform independent shell customization, at GSI, the default for `ENV` is set to

```
/usr/local/bin/.kshrc
```

This common shell script, in turn, invokes your 'private' shell script `~/.kshrc`, if existing. Therefore you shouldn't overwrite the value of the environment variable `ENV`, because the common shell customization would be unavailable for you.

HOME: The default directory after login. You switch to the home directory, when you specify the command `cd` (change directory) without options. It is set by the system to

`/u/`*username*

LPDEST: The default printer (default: pshp9ad).

PATH: Defines the search path for the shell when looking for commands in the system file structure, which is different in different Unix flavors. For example, in HP-UX the PATH variable has by default the value

```
/bin:/usr/bin:/usr/contrib/bin:
/usr/local/bin:/usr/bin/X11:
/etc:$HOME/bin:.
```

The directories in the path are separated by colons (:). The search order is from left to right. The environment variable `HOME` contains the value of the **home directory**, the default directory after login (see above).
The last directory in the PATH variable, indicated by '.', specifies the current working directory. If you have a lot of own commands and wish to put your current working directory to the first position in your search path, specify

```
$ export PATH=.:$PATH
```

PS1: The default command prompt in your shell. At GSI, it is set to

hostname`:/u/`*username sequencenumber*`$`

or, to give an example:

```
rzri6f:/u/goeri 33$
```

TERM: The terminal type for which output should be prepared. Depending on the Unix flavor, **aixterm**, **hpterm**, or **dxterm** are assumed as default for AIX, HP-UX, or ULTRIX systems, respectively. If necessary, you should overwrite it with **vt100**, **vt200**, **3270**, and so on, depending on your terminal type.

VISUAL: Is set to the value **emacs** to enable command line editing with emacs syntax (see section 5.2.6 on page 31.

The Profile Files

Using the Korn shell, there are normally four files for the customization of the environment:

```
/etc/profile
~/.profile
/usr/local/bin/.kshrc
~/.kshrc
```

If using HP VUE, there is an additional file `~/.vue-profile` available. The files in the users home directory are available for private customization. The two other files are available for common customization on all GSI platforms and can only be modified by the corresponding system manager. The file `/usr/local/bin/.kshrc` can only be utilized, if the ENV environment variable is set appropriately (see the description above).

The organization of profile files in normal Korn shell environments and in HP VUE environments is a little bit different. Therefore, at the HP workstations, the file **`~/.profile`** is split into two files:

```
~/.profile
~/.profile-common
```

The file `~/.profile-common` contains the customization settings that are the same for both environments.

1.7.2 Command Line Editing

Command line editing can be done with emacs syntax (see section 5 on page 28. Available functions are:

<Ctrl-p>	get previous command from history file
<Ctrl-n>	get next command from history file (requires at least one <Ctrl-p> in advance)
<Ctrl-b>	move cursor backwards in command line
<Ctrl-f>	move cursor forwards in command line
<Ctrl-d>	delete under cursor
<Ctrl-a>	jump to begin of command line
<Ctrl-e>	jump to end of command line

A mask for the next command to be executed can be obtained with <Ctrl-p> or <Ctrl-n>. The cursor is moved within the command line with <Ctrl-b> or <Ctrl-f>. At any position, characters can be inserted, or be deleted with the <delete> key, with <Ctrl-h>, or <Ctrl-d>. The command history is accessed from the file `~/.sh_history`.

Chapter 2: UNIX Commands

2.1 Constructing a command line

In general, a command line consists of three parts, although not every command requires all three parts:

general Unix Command

```
Name of Command  Command Options
                       Name(s) of File(s)
```

There isn't much to say about the command's name, except that most UNIX commands have short names. Command options are usually designated by a hyphen (or minus sign), followed by a single letter also called a **switch**. Sometimes you can type more than one letter after a single minus sign to indicate multiple options; sometimes you cannot. In a few instances, command options are designated by plus signs instead of minus signs. Many commands allow one or more input files to be named. Output files are generally, but not always, designated by an output option switch like `-o`. Another method of designating an output file will be discussed later in this chapter. The various options and filenames that follow the command are referred to, collectively, as **arguments**.

An Example

Consider the `ls` command, discussed in Chapter 3. The UNIX `man` pages, an online help facility in the UNIX environment, show 21 possible options for this command:

`ls` command options

```
Name    Command Options           File(s)
ls      [-RadCxmlnogrtucpFbqisf]  [ name...]
```

In case of this command, there are 22 different options - 21 switches plus one no switch - you can use. The brackets which are not to be typed on the command line, indicate that all option switches are optional, never required. More than one option can be typed after a single minus sign though. Finally the word `name` indicates that you can type at least one directory or file name after the options. See the UNIX online manual command `man` for further information.

2.1.1 Redirection of Input and Output

Unix regards the terminal's keyboard as its **standard input: stdin(0)** and the terminal's screen as **standard output : stdout(1)**. However, with most UNIX commands it is possible to **redirect** the input and output of a command. The symbols used in a command line to request redirection are the **less sign (<)** and the **greater sign (>)**.

Redirection of Input

In case you want to use a file as input to your program, type

```
prog < file
```

Redirection of Output

In case you want to redirect the output of your program to a file, type

```
prog > file
```

To append the output to an existing file rather than to overwrite it, use the redirection symbol `>>` :

```
prog >> file
```

To suppress output redirect it to the null device like :

```
prog > /dev/null
```

2.1.2 Pipelines

In addition to redirection of input and output, UNIX can connect two processes with a **pipe**, so that the output of the one process becomes the input for another. The symbol for a pipe is the **vertical bar** (|). It is possible to set up multiple pipelines. Commands that appear in pipe statements may include all the usual options and file designations.

```
man ls | lp
```

This command pipes the output of the `man ls` command to the printer via the `lp` command.

2.2 Regular Expressions

When file and directory names are used you can specify some special characters as pattern that the shell matches against the file names in a directory. These special **pattern-matching characters** are :

`*`	Matches any string, including the null string
`?`	Matches any one character

`[...]` Matches any one of the characters enclosed in square brackets

`[!...]` Matches any character other than one of the characters that follow the exclamation mark within square brackets.

Inside square brackets, a pair of characters separated by a `-` (hyphen) specifies a set of all characters lexically within the inclusive range of that pair, so that `[a-dy]` is equivalent to `[abcdy]`.

Using pattern-matching characters in file names on the command line has some restrictions. If the first character of a file name is a `.` (dot), it can be matched only by a pattern that begins with a dot. For example, `*file` matches the files `myfile` and `yourfile` , but not `.myfile` or `.yourfile`. Use the pattern `.*file` to match these files.

The character `@` at the end of the pattern is ignored during the matching. However, the `@` is appended to the corresponding component of the matched file names to allow hidden directories to be referenced directly.

If the pattern does not match any file names, the pattern itself is returned as the result of the match.

2.3 Quick Reference of Commands

This reference summarizes frequently used UNIX commands, special characters used to identify directories, and special characters used on the command line. More detailed information on each command, including a complete list of options, can be obtained with the `man` command.

2.3.1 Managing Directories

`pwd`	display the path name of the working directory
`cd` *dir*	change working directory to *dir*
`mkdir` *dir*	create directory called *dir*
`rmdir` *dir*	remove (delete) directory called *dir*. *dir* must be empty

2.3.2 Managing Files

`ls`	list contents of working directory
`ls` *file*	list *file* if it exists in working directory
`ls` *dir*	list contents of the directory *dir*
`ls -l`	list additional information on directory contents
`ls -a`	list all files including hidden files
`cp` *file1 file2*	copy *file1* to *file2* (overwrites *file2*)
`cp` *file dir*	copy *file* into directory *dir*
`mv` *file dir*	move *file* into directory *dir*
`mv` *file1 file2*	move *file1* to *file2* (overwrites *file2*)
`rm` *file*	remove (delete) file
`rm -i` *file*	ask for confirmation before removing (deleting) *file*
`more` *file*	displays contents of *file*, one screen at a time
`cat` *file*	displays contents of *file*
`chmod` *arg file*	change read/write/execute permission of *file*
`chmod` *arg dir*	change read/write/execute permission of *dir*

2.3.3 Managing Jobs

`<Ctrl-c>`	stop current job
`ps`	list process by process identifier
`ps -fu` *user*	full listing of all processes of user *user*
`kill` *PID*	stop process with process identifier *PID*
`jobs`	list your jobs by job number

2.3.4 On-line Help

`man` *command*	display manual entry for *command*
`man -k` *keyword*	list manual pages that pertain to keyword
`learn`	on-line tutorial (AIX only)
`info`	on-line tutorial, help pages and manuals, stored on CD-ROM. Can only be used with X-Windows Terminals.

2.3.5 System Information

`who`	list users logged onto system
`who am i`	displays your logon ID
`finger` *user*	displays information on *user*
`passwd`	change password

2.3.6 Utility Programs

`sort file`	sort contents of *file*, send result to standard output
`grep` *pattern file*	look for *pattern* in *file*
`uniq` *file1 file2*	delete repeated lines in *file1*, write new version to *file2*

`wc` *file*	count the number of lines, words, and characters in *file*
`echo` *string*	write Paramstring to standard output, translate special characters
`find` *path*	search the directory tree `path` (see `man` page for more details)
`tar` *file*	write to or retrieve files from an archival storage media
`compress` *file*	compresses the file and writes *file*`.Z`
`uncompress` *file*`.Z`	restores *file* from compressed file

2.3.7 Directory Identifiers

~	your home directory
`.`	the working directory
`..`	the parent directory (one level up within hierarchy)
`/`	root directory

2.3.8 Special Characters

*	match any character
<	redirect standard input
>	redirect standard output
|	send standard output of first command to standard input of second command (pipe)
&	put job in background

2.4 Shell Scripts

The commands described in the previous examples can be executed when typed at the command prompt, but they can also be written to files and executed as command lists or **shell scripts**, which correspond to TSO CLIST in MVS or DCL in VMS. For example, if you have a command list in a file named `myscript` in the current working directory, then just enter

```
myscript
```

at the command prompt. The command list will be executed. If not, you should check two things at first:

1. Is the file containing your shell script is **marked as executable**? See section 3 on page 14 for more information!
2. Look with the `ls` command if the required file really exists in your current working directory!

If the file resides in another directory than your current working directory, and if you don't want to switch there, specify either the full (**absolute**) path name or the path name **relative** to your current working directory. Let's assume, for example, your script resides in /u/goeri/util, and your current working directory is /u/goeri. Then you can enter either

```
util/myscript
```

using the relative path name (with no / at the beginning), or

```
/u/goeri/util/myscript
```

specifying the absolute path name (with / at the beginning).

If you define **environment variables** (see section 1.7.1 on page 8) in your script, you should invoke it with a preceding dot, if the environment variable shall be valid in the current shell:

```
. myscript
```

If not, the environment variables are valid only within your script and all subprocesses invoked from there.

Chapter 3: Files and Directories

3.1 The UNIX File System

UNIX has a structured filesystem that contains three kinds of files:

directories which store the names of other files including other directories;

ordinary files which store text, source programs, and object code; and

special files which correspond to peripheral devices.

3.1.1 Naming Directories and Files

The **root** directory is identified by a single character: slash (/). To name one of the major directories directly under root, type slash (/) to represent root, followed by the directory's own name, as in `/usr`. The slash in front of `usr` tells you that usr is a subdirectory of root. An example of a typical Unix file system is shown in figure 3.1 on page 15.

`/u` user directory
`/bin` binary directory
`/dev` device directory
`/etc` miscellaneous directory
`/tmp` temporary directory

To identify the user's home directory, type another slash after `/u`, followed by the account name, as in `/u/otto`. The first slash refers to the root directory, `u` identifies the **parent directory**, and `otto` is a **subdirectory**.

3.1.2 Rules for Naming and Accessing Files

The rules for naming and accessing files and directories are closely related to the structure of the UNIX file system:

- The root directory is identified by a slash (/).
- A simple filename can be any combination of 1 - 14 characters **other than** slashes (/), asterisks (*), question marks (?), quotation marks (") or ('), square brackets ([) or (]), dollar sign ($) or control characters.
- A path name is a sequence of directory names, possibly followed by a simple filename, with each pair of names separated by a slash (/).

To avoid misinterpretation, the safest characters to use for simple filenames are letters of the alphabet, numbers, periods (.), hyphens (-), and underline (_). Note: in UNIX, upper and lower case are **not** the same !

The directory permanently assigned to you is called your **home directory**; this is the directory to which you log on. Any directory to which you may move after logging on (including your home directory) will be called your **current directory**, or **working directory**, as long as you remain in that directory. The directory which is one level above your current directory in the file system is called your **parent directory**. UNIX provides shorthand symbols to indicate your current directory (.) and your parent directory (..). If a path name used to access a file begins with a slash (/), then the search for the file begins at the **root directory**. Such a path name is called an **absolute path name** or **full path name** . If a path name begins with a simple filename, then the search for the file begins at your current directory. Such a path name is called a **relative path name** .

3.2 Working with Directories

3.2.1 Displaying the contents of a directory: `ls`

To sort and display the names of all the directories and files that reside in you current directory, use the `ls` command:

```
$ ls
file1
file2
file.3
Mail
$
```

3.2.2 Changing the Working Directory: `cd`

To change your working directory, that is to move to another directory, use the `cd` command:

```
$ cd /u/otto/Mail
$
```

3.2.3 Determining Your Working Directory: `pwd`

To find out the name of your working directory at any moment, use the `pwd` command:

```
$ pwd
/u/robin
$
```

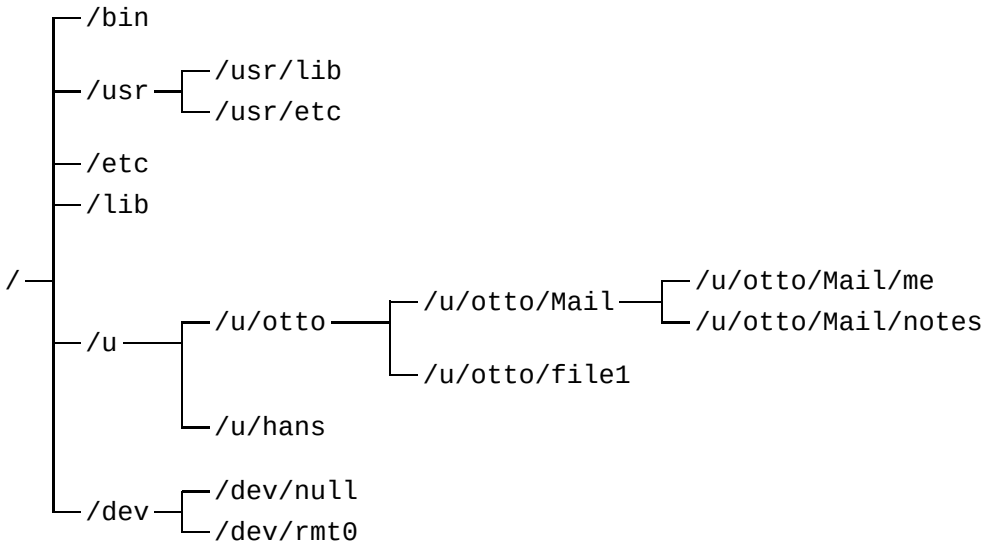


Figure 3.1: Part of a typical UNIX file system

3.2.4 Creating a New Directory: `mkdir`

To create a new subdirectory within your current working directory, use the `mkdir` command:

```
$ mkdir messages
$
```

This command will create a new subdirectory called messages.

3.2.5 Removing an Existing Directory: `rmdir`

To remove an existing directory from your working directory, move to the target directory, delete all its files, move back to the parent directory, and then use the `rmdir` command:

```
$ cd /u/useless
$ pwd
/u/useless
$ rm -i *
$ cd ..
$ rmdir useless
```

If you try to remove a directory that is not empty, you will see a warning displayed. You may use the following shorter method instead of the above:

```
$ rm /u/useless/*
$ rmdir /u/useless
```

or:

```
$ rm -r /u/useless
```

3.2.6 Renaming a Directory: `mv`

To change the name of a directory, use the `mv` command:

```
$ mv old.name new.name
```

3.3 Working With Files

3.3.1 Displaying the Contents of a File: `cat`

To display the contents of a file, use the `cat` command. It simply displays the contents of the file or several files on the screen (standard output):

```
$ cat file.3 file.1
```

Combining Files

Another function of the `cat` command is to combine files, or concatenate files with the result stored in another file e.g.:

```
$ cat file.1 file.2 > file.3
```

Avoid storing the result in one of the original files, since this will the original file to be overwritten.

3.3.2 Renaming a File: `mv`

You can use the `mv` command to rename a file or to move it from one directory to another. To change the name of a file, enter a pair of command like this:

```
$ cat new.file
cat: cannot open new.file
$ mv old.file new.file
```

The `mv` command will change the file's name whether the new filename exists or not. The `cat` command makes sure that a file will not be replaced and be lost.

3.3.3 Copying a File: `cp`

To make a duplicate copy of a file, use the `cp` command:

```
$ cp file.one FILE.ONE
```

This command will make a copy of file.one. As a reminder lower and capital letters are **different** filenames.

3.3.4 Deleting a File: `rm`

To delete a file, use the `rm` command:

```
$ rm file.1
```

This form of the command will delete the file file.1 immediately. To confirm before proceeding to delete the file, add the **-i** option:

```
$ rm -i file.1
```

3.4 File and Directory Permissions

UNIX allows you to access other files and directories in the system, but only if you have permission from the owner of those directories and files.

3.4.1 Determining Permission: `ls -l`

To determine the permission associated with a given file or directory, use the `ls -l` command to display the contents of the directory:

```
$ ls -l
total 501
-rw-r-----
1 user group  108 Oct 15 19:10 file.1
-rwxr-x---
1 user group 6452 Oct 15 17:15 program.1
drwxr-xrw-
1 user group  512 Oct 15 19:13 letters
```

The first character indicates the type of the file

```
-  ordinary files
d  directory
l  links
```

The remaining nine characters represent three sets of three characters: one set for the individual user, one for the user's working group, and one for all other users. Spread out the characters of the display above to explain the groupings:

```
Type   User   Group   Others
-       rwx   r-x     ---      program.1
d       rwx   r-x     rw-      letters
```

The permissions given are for **reading**, **writing**, and **executing**. They have different meanings for ordinary files and directories. For an **ordinary file**, permissions are defined as follows:

read permission means you may look at the contents of the file

write permission means you may change the contents of the file

execute permission means you may execute the file as if it were a UNIX command.

For a **directory**, permissions are defined as follows:

read permission means you may see the names of the files in the directory

write permission means you may add files to and remove files from the directory

execute permission means you may change to the directory, search the directory, and copy files from it.

The characters used to represent these permissions are:

```
r     read permission
w     write permission
x     execute permission
-     permission denied
```

3.4.2 Changing Permission: `chmod`

You can make changes to permissions by entering a `chmod` command. It allows the owner of the file to add to (+) or remove from (-) existing permissions. It also allows the owner to clear existing permission and assign all permission from scratch; this is known as assigning permissions absolutely (=). The `chmod` command affects any of the three types of access for any of the three categories of UNIX users, using one-letter symbols in the following order (left to right):

```
u       owner (user)
g       File's group
o       all others
a       all (default)

+       add permission
-       remove permission
=       absolute permission

r       to read
w       to write
x       to execute
```

Caution: It is possible for you to lock yourself out of one of your own files with `chmod`. Be careful when you type it.

Example:

```
$ ls -l
-rwxr-xr-x 1 otto rz 487 Jul 30 10:21 psab
$ chmod o-x psab
$ ls -l
-rwxr-xr-- 1 otto rz 487 Jul 30 10:21 psab
```

In the above example, the first `ls -l` shows the default permissions for a script, which is executable and readable by everyone, but writable only by the owner. After the `chmod o-x` command, the execution permission for others is removed.

Chapter 4: Basic Services

In this chapter we assume that the reader has successfully logged in, is somewhat familiar with the basic Unix commands and understands the fundamentals of the Unix file system. We explain here the use of some everyday infrastructure services such as getting help, mailing, printing, remote login, remote execution and the backup/recovery mechanism. Editors are not mentioned in this chapter but can be found in chapter 5 on page 28. If you are interested in text processing and page layout please refer to chapter 6.1 on page 34

4.1 Help and Documentation

4.1.1 How to get Help

To get online help in a Unix session, there are in principle three methods available:

- command `man`
- command `info` (on AIX/6000 and HP-UX platforms only, with graphics terminal only).
- help button in HP-VUE environment (HP-UX only)

4.1.2 The `man` Command

The `man` command is the standard Unix command to get online command descriptions. Specifying for example

```
man man
```

gives the first page of the description of the `man` command on your screen. Following pages are displayed after pressing the `<Enter>` key. Output can be interrupted with `<Ctrl-c>`.

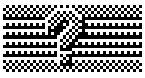
4.1.3 The `info` Command

The complete manual information available on a CD-ROM can be obtained in a very comfortable environment with several screens using the `info` command. The implementations on AIX and HP-UX are similar in functionality, but different in the layout of the screens and menus. Both systems provide extensive help menus on functionality and usage and are rather self-explaining.

In principle, the following functions can be performed:

- Any manual page can be **browsed** which allows to read the manuals online.
- All or parts of the manuals can be scanned for **keyword search**. In this mode, a specific text expression can be entered to be searched for. If found, the number of occurencies will be indicated, and the corresponding manual parts are located and offered for reading.
- On the RS/6000, the AIX command descriptions can be accessed via menu offering the first letters and command categories.

4.1.4 The VUE Help button



In the HP VUE environment you can press the help-button (marked by a question mark (?) in the lower row of the HP-VUE workspace manager) and you will get a help-window with information on the following topics, which can be selected by a mouse-click on the appropriate line:

- Tutorial on HP VUE
- `man` pages
- Help on Customization of HP VUE
- Fortran and other applications
- . . .

4.2 Mail Facilities

There are several different kinds of mailers available on our unix systems. AIX, HP-UX, and ULTRIX provide the standard `mail` or `mailx` command. In addition there is the public domain `elm` mailer, which allows a comfortable and easy to use mail handling.

Mail addresses are of the general form *user@node.domain*. For a few examples of valid mail addresses see the description of the **$HOME/.mailrc** alias definition file below. You can send mail to any internet-, BITNET-, and GSI-DECnet-nodes. When mailing to BITNET use addressing of the form *user@node*`.bitnet`, for GSI DECnet nodes use **v6000a** as gateway.

4.2.1 The Standard Mailer

Example: mailx on AIX:

Send a mail message:

`mailx username` or `mailx aliasname`

Type the message text, terminate and send your message with CTRL-d or with a last line containing a single . period in column one.

Aliases are defined in the **$HOME/.mailrc** file. This file may possibly contain lines like the following.

```
alias hd     dob@hp9a.gsi.de
alias um     rz02@mvs.gsi.de
alias rb     brun@cernvm.cern.ch
alias he     goofy@v6000a.gsi.de
alias body   user@cageir5a.bitnet
alias group  dob,goeri,schwab
```

Aliases may contain lists of addresses but not lists of aliases.
Mail a file:
`mailx` *user* `-s "next meeting"< dates.txt`

Reading, forwarding, replying, filing, sorting and editing mail are done inside the mail utility. Invoke:
`mailx`
then use any of the following subcommands:

m	send mail, invoke editor
h	display list of messages in your mailbox
?	help
d	delete current message
e	edit the current message
[n]	read message number [n]
-	read previous message
s	save current message in personal mailbox
s file	save current message to file
s [n] file	save message number [n] to file
c [n][file]	same as s but do not delete message from incoming mailbox
r	reply to current message
a	display aliases
a hd	display alias hd
q	quit mail, discard deleted messages
x	quit mail, do not discard deleted messages

By default mail uses the vi editor. Insert a line
`set EDITOR=/usr/local/bin/emacs`
into your **$HOME/.mailrc** file to change the default editor to emacs.
If you want to work with your personal mailbox instead of the system mailbox, type
`mailx -f`
Work with an arbitrary mail folder is started as
`mailx -f filename`
You can forward all mail arriving for you on some node which you do not use frequently by creating a file **$HOME/.forward**. This file should contain a line *user@othernode*, e.g.
`otto@rzhp9a.gsi.de`
This will automatically forward all incoming mail to *user@othernode*.

Mail on Ultrix works very much the same way. Invoke mail with command `mail`, however.

4.2.2 The `elm` Mailer

`elm` is a screen-oriented electronic mail processing system. In interactive use, the main header index and mini-menu of commands are displayed upon initial invocation and at any point when the program is waiting for input.

You can also invoke `elm` by the mail button of the HP-VUE workspace manager (upper row, 4th left button).

You can easily send a mail (`m`), reply (`r`) or forward (`f`) a mail, save it to a folder (`s`) or delete an incoming mail (`d`). By command `a` you enter the alias menu.

Aliases are stored in the file `~/.elm/aliases.text`, which can be edited by any text editor. After having edited this file, the internal alias file has to be updated by the command `elmalias`. In elm aliases may also contain a list of aliases.

4.3 Local Networking Tools

In a network more or less often you will encounter the situation that a command or file is not available on your local node. You will then want to work on a remote host (node) or transfer files between any two nodes of the network.

Some of the commands described below do only work between *equivalent* UNIX hosts. This applies to `rlogin`, `remsh`, and `rcp`. The effect users see on equivalent hosts is that when they are logged on to one of these equivalent hosts, they can work on all equivalent hosts with the above commands without further authentification. Equivalent hosts at GSI are currently: `rzri6a`, `rzri6f`, `rzhp9a`, `rzhp9b`, and `rzds5a`.

The equivalence of hosts is maintained by the administrators of these hosts.

Other commands work on/between arbitrary hosts with almost any Operating System. To run such a command the host must be connected to the IP net and therefore be able to handle the TCP/IP protocol. These commands, namely `telnet` and `ftp`, allow much more connectivity but — of course — less transparency.

4.3.1 Remote Processing

There are several ways to do some work on a remote host without ending the local session. With the commands

`telnet` and `rlogin`

you can establish a session on a remote host from within your local session. Whereas the commands

`remsh` and `rexec`

do not perform a login on a remote host but execute commands there for you.

`telnet`

To enter the `telnet` environment simply issue the command

`telnet`

and after telling you the escape character (usually `<Ctrl-]>`) `telnet` will show its prompt

`telnet>`

so now you can enter `telnet` subcommands.

For a complete list of subcommands and flags of the `telnet` command consult its man page. Here are some often used subcommands:

`quit`	ends the `telnet` command
`open`	establishes a connection to a remote host
`close`	ends that connection
`help`	lists the subcommands with a brief explanation

So the first subcommand will probably be:

`telnet> open` *hostname*

This connects your terminal or X-window to the specified host and displays the logon logo. So you can log on there and work as usual. After logging out of the remote host you will get the

`telnet>`

prompt so that you can `open` the next host or `quit` from the `telnet` program an resume the local session.

Alternatively you can specify the remote host on the invocation of the `telnet` program:

`telnet` *hostname*

This automatically opens the remote host and you see immediately the logon logo. Logging out of the remote host will now end the `telnet` program. You don't get the `telnet>` prompt, you are back in your local session.

`rlogin`

The "remote login" command

`rlogin` *hostname*

connects your current terminal or X-window to a login session on the specified host. Since your local host must be equivalent (see above) to the remote host you do not need to authenticate yourself to the remote system. You will get the command line prompt of the remote system. Logging out there resumes your current local session.

Usage Notes:

You must not omit the *hostname* parameter when using `rlogin`.

If you use flags on this command, you have to place them **after** *hostname*!

`remsh`

To execute a command on a remote host enter:

`remsh` *hostname command*

This command runs a "remote shell" which executes *command* for you. You can *not* work interactively on the remote host, all input to the command must be specified on the local command line, all output from the command is directed to the local standard output. I/O-redirection works as follows: Using the `>`, `>>`, `<`, `<<` operators as usual redirects in- and output to and from the remotely executed command to *local(!)* files. If you want to redirect the in- and output to *remote* files use double quotes `"` around the redirection operators.

Usage Notes:

On AIX `remsh` and `rsh` are synonyms.

The `remsh` command will *not* process the login profiles, but the `$ENV` file (usually the `.kshrc`). If you omit the *command* parameter on the `remsh` command the `rlogin` will be executed instead, which — of course — *will* process the login profiles!

If you use flags on this command, you must place them **between** *hostname* and *command*!

Examples:

All the examples below assume, that you are logged on to a host that is equivalent to `rzri6f`:

Get a window with a terminal emulation on your X-server:

```
remsh rzri6f aixterm -display $DISPLAY &
```

To get a window that emulates a mainframe terminal (3270) on your X-server enter:

```
remsh rzri6f x3270 -display $DISPLAY  mvs  &
```

The next examples do only illustrate the I/O redirection mechanism with the `remsh` command. They are *no* examples for effective file transfer (see below!).

```
remsh rzri6f cat .profile >> .profile
```

appends the profile you have on the `rzri6f` to your *local* profile.

Find the difference:

```
remsh rzri6f cat .profile ">" .profile.old
```

Right! Your remote profile will be copied to a remote file named `.profile.old`.

`rexec`

With the `rexec` command you can execute a command on every remote UNIX host you have a account on regardless if they are equivalent or not. The `rexec` command works just in the same fashion `remsh` does, the only difference is that you will be prompted for your name and password on the remote host.

`rexec` *hostname command*

You can try the examples from the last section “remsh” with one pitfall: If you use unquoted I/O redirection the authentification prompts do not work properly, you will not see the prompts, but you can type in your username and password “blindly”.

4.3.2 File transfer

Let us focus on the two commands

`rcp` and `ftp`

which allow file transfer between the nodes of a network.

`ftp`

The `ftp` (file transfer protocol) allows you to login to a remote node and execute `ftp` subcommands there without leaving your current session on the local host. The `ftp` command works between various platforms, not only between UNIX systems. Simply invoke `ftp` by typing

`ftp`

and get the prompt

`ftp>`

Select the remote host by

`ftp> open` *hostname*

and you will be prompted for login. If you invoke ftp by

`ftp` *hostname*

the `open` *hostname* is implicitly executed and you will be directly prompted for login. For the various flags that can be set on the command line when invoking `ftp` please consult the man page.

After a successful login you will get the `ftp>` prompt again and you can now issue the `ftp` subcommands which allow you to navigate through the remote filesystem, display the remote directory and — of course — transfer files between the remote and the local host in both directions. Some often used `ftp` subcommands are:

`quit`	ends the `ftp` command
`cd`	changes directory on remote host
`lcd`	changes directory on local host
`mkdir`	creates a new directory on the remote host
`pwd`	prints the path that is current on the remote host
`put`	transfers a file from local to remote
`get`	transfers a file from remote to local
`binary`	the data are transferred without conversion
`ascii`	the data are converted due to different character representation on the sending and receiving host.

`help` displays *all* available subcommands and gives a short description of them.

You find more information on the very complex `ftp` command in the man pages, info systems, and TCP/IP manuals.

Usage Notes:

Some explanations on data conversion: If you transfer a file that is human readable you will do it in `ascii` mode, this is the default. This is the right mode to transfer e.g. a TEX source. To transfer compiled programs or similar files you have to switch to `binary` mode, so the transfer will be done on a "bit by bit" mode without any change. This is the right mode to transfer e.g. a `dvi` file produced by the TEX program.

Examples:

Imagine you have to be up to date on remote files which change frequently. You will have to perform the same file transfer quite often. A shell script similar to the following will be very useful then:

```
# getfiles: get a few files regularly from MVS
#
# customize the next lines:
user=XY12        # replace this with your id
locpath=...      # specifies where to put
rempath=...      # from where to get
#
echo " "
echo "Opening FTP connection to MVS"
echo "for user" $user
echo " "
ftp -n mvs <<EOF # invoke FTP with
                 # next lines as input
user $user       # specifies remote user id
lcd $locpath     # set the local path
cd $rempath      # set the remote path
get file1   # with EBCDIC/ASCII conversion
binary
get file2   # without conversion
quit             # terminate FTP
EOF              # terminate input to FTP
```

You will be prompted for your password on the remote host.

rcp

The `rcp` (remote copy) command copies a file or directory from one host in the network into a directory or as a file on another host of that network:

`rcp [-rp]` *source destination*

Remember that the hosts have to be equivalent (see above)! The `-r` flag means *source* is a directory and is to be copied with all the files and subdirectories it contains, the `-p` flag preserves the modification time and access modes.

The *source* and *destination* specifications do not only contain the name of the file but also the host where it resides. You can specify *source* and *destination* in one of the following three forms:

filename	relative or absolute name of local file
host`:`*filename*	absolute name of file residing on *host*
user`@`*host*`:`*filename*	name of file relative to the home directory of *user* on *host*.

Usage Notes:

You can use the `rcp` command to transfer files between hosts that are *not* equivalent if and only if the owner(s) of the remote file(s) give you their permission explicitly in a file named `.rhosts` in their home directory or if the remote account(s) do(es) not require a password.

Examples:

To copy the file `some.data` from the current directory of the local host to the directory `/u/hugo/archive` on host `rzri6f` enter:

```
rcp some.data rzri6f:/u/hugo/archive
```

Suppose you want to choose a different name for the *destination*:

```
rcp some.data rzri6f:/u/hugo/archive/x.y
```

You could have specified the *destination* relative to hugo's home directory:

```
rcp some.data hugo@rzri6f:archive
```

This is absolutely equivalent to the first example.

A last and quite complicated example (to be entered on one line):

```
rcp -rp hinz@rzhp9a:measurements/data
rzhp9b:/exp.data/march
```

Here both `rzhp9a` and `rzhp9b` are remote nodes. The *source* is given relative to the home directory of hugo on `rzhp9a` whereas the *destination* is an absolute pathname on `rzhp9b`. The flags say, that the *source* is a directory and is to be copied recursively (`-r`); this implies that the *destination* also has to be a directory. Moreover the file permissions and the modification times are preserved (`-p`).

4.4 IBM Mainframe Access

To access mainframe applications, which require fullscreen support (e.g. ISPF on the MVS system), the terminal client software on the UNIX system must provide the emulation of a IBM 3270 terminal.

4.4.1 Alphanumeric sessions

tn3270

The `telnet` environment on the IBM RISC System/6000 workstations offers in addition to the line-by-line mode the emulation of a DEC VT100 terminal or an IBM 3270 terminal. Also public domain `tn3270` software is available, e.g. NCSA-Telnet for DOS PC's, which includes also a Tektronix graphics emulation. To start the telnet 3270 emulation on AIX, enter

```
telnet -e 3270 mvs
```
or simply
```
tn3270 mvs
```

which connects directly to the MVS system. The `telnet` command supports some standard 3270 terminal types with different lines and columns. The maximum screen size will be used, depending on the lines/columns values of the screen, from which the session is initiated. The 3270 keyboard mapping is determined by the following precedence:

`$HOME/.3270keys` specifies the user's keyboard mapping and overrides the system defaults. `/etc/3270keys.hft` system default keyboard mapping for use with standard workstations or consoles. `/etc/3270.keys` system default keyboard mapping for use with limited function terminals. On a color display, e.g. a X11 screen, the colors and field attributes are displayed the same as those of an IBM 3279 terminal.

4.4.2 Access with graphics capabilities

x3270

`x3270` is a X Windows system based 3270 emulator. It provides 3270 terminal sessions for UNIX workstations, accessing an MVS host system. GDDM type graphics is supported, when specified by the corresponding flags during session initiation. `x3270` is installed on the IBM RISC/6000 systems RZRI6F, RZRI6A and RZRI6B. Two shell scripts provide for easy access to MVS via `x3270` without the need to specify flags, fonts, colors etc.:

`mvs` for use at workstations with IBM (MF) keyboard

`mvsvt` for use at workstations with VT100 keyboard

For extended information about `x3270` options and parameters type

```
man x3270
```

GDDMXD

`GDDMXD` is an interface between the MVS GDDM graphics system and workstations, supporting the X Windows system. The GDDM data stream created by the MVS application (e.g. SATAN, GNOM, DCF) is translated to the X Windows system protocol and transmitted by TCP/IP to the X Windows server for the display. `GDDMXD` can be used from any workstation or even PC with X-Windows support without the need to login to an UNIX account at a GSI workstation and without the need to take care for required X11 fonts.

To initiate the GDDMXD output, you have to enter the TSO command

```
GDDMXD ON
```

outside ISPF. The TCP/IP address of the target display is read from the MVS dataset `user_id.XWINDOWS.DISPLAY`. Options for the grafic window on the X station like screen size, geometry, color table etc. can be modified by means of the dataset `user_id.X.DEFAULTS`. During the session, graphics output must be acknowledged on the X screen by hitting any PF-key, otherwise the host application will not continue. For more information contact users help desk.

4.5 Print Services

The printers available to the unix machines are listed in table 4.1. We support printing of plain text ASCII files and PostScript documents. The widespread LN03 printers are accessible from Unix. For large documents we recommend to use the higher throughput **pshpa** and **psaa** printers in the computer center's output room 2.223. Printers in this room are maintained by central operations (phone 515). Printer **psteka** provides support for color slides. Special transparencies are required, they are available upon request from the operators. For the time being use the manual feeder for slides on **psteka**, you need some patience. Printing costs approximately DM 4.- per color slide.

Name	Location	Model	Options	reachable from
pshpa	I/O Room RZ 2.223	HP LaserJet IIIsi	A4 B/W ASCII, PS, PCL-5 single/double-sided	DEC, HP, IBM
pshpad	I/O Room RZ 2.223	same as above	A4 B/W ASCII, PS, PCL-5 double-sided	DEC, HP, IBM
psaa	I/O Room RZ 2.223	DEC LPS20 Printserver	A3, A4 B/W ASCII, PS single/double-sided	DEC, HP, IBM
psteka	I/O Room RZ 2.223	Tektronix Phaser 3 Pxi	A3, A4 Color PS Transparencies	DEC, HP, IBM
ln03r_a	Elex Lab 2.252 2.252	DEC LN03R	A4 B/W ASCII, PS	HP
ln03r_c	KP3 3rd Fl. 4.174	DEC LN03R	A4 B/W ASCII, PS	HP
ln03r_d	Cave-B Meßhütte	DEC LN03R	A4 B/W ASCII, PS	HP
ln03_a	ElEx Lab 2.252	DEC LN03	A4 B/W ASCII	HP
ln03_c	Meß-Station 1.124	DEC LN03	A4 B/W ASCII	HP
ln03_d	AP 1rst Fl. vis-a-vis 2.294	DEC LN03	A4 B/W ASCII	HP
ln03_e	KP2 3rd Fl. vis-a-vis 4.141	DEC LN03	A4 B/W ASCII	HP
ln03_g	SHIP	DEC LN03	A4 B/W ASCII	HP
ln03_j	Cave-B Meßhütte	DEC LN03	A4 B/W ASCII	HP
ln03_k	FRS Meßhütte	DEC LN03	A4 B/W ASCII	HP
ln03_l	ESR Meßhütte	DEC LN03	A4 B/W ASCII	HP
ln03_m	Cave-C Kaos Meßhütte	DEC LN03	A4 B/W ASCII	HP
ln03_n	Cave-A Meßhütte	DEC LN03	A4 B/W ASCII	HP
ln03_o	KP2 3rd Fl. vis-a-vis 4.141	DEC LN03	A4 B/W ASCII	HP
ln03_p	Sicherheit & Strahlenschutz	DEC LN03	A4 B/W ASCII	HP

Table 4.1: List of available Printers

Printing commands vary slightly from platform to platform. The principal commands are shown here in tabular form, for more detail see the examples below.

	Print	**Status**	**Cancel**	**Env.Var.**
DEC	`lpr`	`lpq` `lpstat`	`lprm`	PRINTER
HP	`lp`	`lpstat`	`cancel`	LPDEST
IBM	`lpr`	`lpq` `lpstat`	`lprm`	PRINTER

The names of the printers and the names of the respective queues coincide. In case of **pshpa** we have also generated a queue **pshpad** which automatically selects double-sided printing. **pshpad** is also the default print queue on all central Unix systems. The individual user may override the system default printer destination by setting the **PRINTER** (for ULTRIX and AIX) or the **LPDEST** (for HP-UX) environment variable. See Section 1.7.1 on page 8 for more information on shell and environment. If you want to overwrite this system default, just add a line `export PRINTER=psaa` into your `~/.profile` ksh-startup file.

Examples for HP-UX:

```
lp -d ln03_d -n2 -m doc.txt
```

Print two copies (-n2) of **doc.txt** on **ln03_d** and send mail (-m) upon completion.

```
lp -d psaa escher.ps
```

prints file escher.ps on Postscript printer **psaa**.

```
lpstat -d psaa
```

display status of printer queue **psaa**. The output may look somewhat like this:

```
printer queue for psaa
psaa is ready and printing via network

psaa-44  dob  priority ?   from rzhp9a
       escher.ps          25499 bytes
```

```
cancel psaa-44
```

will cancel that particular print job.

```
man lpstat | lp
```

prints the man page for command **lpstat** on the system default printer, in this case **pshpad**.

```
man cancel | lp -d psaa
```

This time it's the man page for command **cancel** which gets printed on the LPS20.

```
XtoPS -screen -monochrome - | lp -d pshpa
```

This is how you dump the contents of your X-window screen to a B/W PostScript Printer, this takes some time and resources

```
XtoPS -frame - | lp -d psteka
```

This is how you dump a single X-window (selected by clicking with the cross-hair cursor) and print it on the color PostScipt printer in the I/O Room. Color screen dumps take even more time and more resources.

Examples for AIX:

```
lpr -d psaa .profile
```

Sends your file `.profile` to printer **psaa**.

Examples for ULTRIX:

Start the **decwrite** utility, pulldown the **file** menu, release on **Print...** , a printer selection panel appears. Select the appropriate printer and click the **OK** button.

This is only a selection of the most frequently used printing options and is far from being complete. For a comprehensive description of the commands we refer the reader to the man pages.

4.6 Backup Services

Backup of disks is an important service, which is offered by the computer center. It should recover from disasters like disk failures, head crashes, etc. as well as from unintended deletion of single files or directories.

All Unix-workstations, which are located centrally in the Computing Center, will be backed up on a regular basis. The following backup-scheme will be performed automatically:

system disks	• monthly full backup • weekly incremental backup • after major updates and changes
user disks `/u/...`	• monthly full backup • weekly incremental or full backup • daily incremental backup
data disks `/d/...`	• no backup
scratch disks `/tmp`, `/s/...`	• no backup • automatic removal of files after 3 days

It is planned to find a common solution for the backup of all workstations at GSI. This system uses the NFS server implementation on the IBM MVS system as the data pool and the MVS data management facilities. Thus an operatorless service will be possible.

However, up to now two different systems are used for the different manufacturers.

4.6.1 IBM RISC System/6000 AIX

Backup strategy

The first implementation of the GSI backup system for all UNIX systems has been done on the IBM AIX RISC/6000 platform. Weekly full backups and daily incremental backups are stored to NFS mounted data volumes on the MVS system. The backup files are managed by the SMS Storage Management System, which allows to use the automatic tape library ATL as backup medium. A user can therefore recover backed up datasets without operator intervention, even at night or weekend. Currently the directory trees `/u/...`, `/usr/local`, `/dk/rzri6b/pub`, `/cern` and `/dk/rzri6b/gnu` of the common file system of the AIX cluster are backed up.

Recover utility

A recover utility has been developed, which allows to recover backed up datasets by request of the owner/user. Currently two full backups will be held on the MVS system and all the incremental backups from the oldest full backup up to the current day. The user can initiate the recover procedure by the command `hrecover`. The current (Julian) day is displayed and the current search options. Enter `o(ptions)` to change the options:

`f(ile)` toggle for file search
`d(irectory)` toggle for directory search
`p(attern)` for a pattern matching file/directory search
`r(ange)` to specify the (Julian) date range
`q(uit)` to accept the specified search options.

Enter `s(earch)` to search for the specified files/directories within index datasets, created by the backup process. The matches are presented to the user, the newest first. You can enter

`s(elect)` to select the file for recovery
`n(ext)` to skip the presented file
`r(ecover)` to initiate the recovery

The backups files on the MVS system are allocated on a per user basis, so the recall itself requires little network load and CPU consumption.

4.6.2 HP HP-UX

Backup

An incremental backup of user files is stored daily at 23:00 on disk, once a week on DAT-tape and a full backup once a month (at the beginning of the month) on DAT-tape. All system disks are backed up weekly and monthly on DAT-tape.

Recovery

Each user can recover his own datasets by the command `recover`. The `recover` command must be started from the machine, where the disks are physically attached (`rzhp9a` for user disks).

As parameter, a regular expression containing the filename and/or parts of the directory name may be given. The recover process searches the indices of all backup files in reverse chronological order and returns the prompt to the user if a matching filename was found, together with time and date of backup. If the user wants to recover this dataset, recovery is started with `y` after the prompt.

Note: Existing datasets are only overwritten, if the date on the backup is newer than the date of the existing file !

As additional features, the user can edit the list of datasets to be selected for recovery (command `e`), search (command `s`) in older backups or abort recovery (command `n`).

Example:

```
rzhp9a:/u/dahlinge 307* recover
which file to restore (reg. expr. possible):
Mail/unix?*
starting : look for pattern Mail/unix?* in
backup Index:

 pattern Mail/unix?* found in backup !
 This backup dates from Oct 9 at  23:01
 The following files have been selected for
 eventual recovery:

-----------------------------------------------
i /dk2/rzhp9a/u/rz/dahlinge/Mail/unix-wizards-
request
-----------------------------------------------

 If you enter y , the above listed files will
                  be recovered.
 If you enter n , recover program will stop.
 If you enter s , the next (older) occurrence
                  of the pattern will be
                  searched
 If you enter e , you can edit the list of
                  files to be recovered
 enter your choice ([n]/y/e/s)y
⋮
```

4.6.3 DEC ULTRIX

Until now, no backup is done.

Chapter 5: Editors

Text editing is one of the important things on any computer system. On a UNIX system several text editors are available. Emacs, vi, ed, LPEX, INed, edt+, uni-Xedit, Vuepad, and xedit are only a few UNIX editors.

A versatile Unix user knows at least vi or Emacs to work come around the different platforms.

The text in this section is designed to get you started using the vi and Emacs editor quickly. For vi and Emacs the elementary editing will be explained.

Before you invoke an editor, you should know one possible pitfall: An editor must know what kind of terminal you are using. If your terminal type is not defined or incorrect, you may get problems with the editor. Quit the editing session and set the correct terminal type (see 1.7.1 on page 8).

5.1 vi Editor

The editor vi is a standard full-screen text editor. and available on nearly all UNIX systems. On AIX, HP-UX and ULTRIX vi is part of the operating system.

The great advantage of vi is, that it is included in the vendor independent international standard POSIX.2 (IEEE 1003.2). On all POSIX.2 conform systems vi is available. This allows users to move from one POSIX system to an other without learning a new editor.

5.1.1 Operating Modes

The vi has three operating modes:

- vi command mode.
- text input (or insert) mode.
- ex command (or line edit) mode.

In the vi command mode, each key initiates a instruction. In the text input mode the keyboard functions like a typewriter. And in the ex command mode you can use the 'old' ex line editor to invoke ex commands.

Like in UNIX all commands in vi are case-sensitive.

5.1.2 Starting vi

To start vi, simply type `vi` followed by the name of the file you want to edit.

`vi` *myfile*

Since it is a new file, the buffer is empty and the screen appears as follows:

```
~
~
~
"myfile" [New file]
```

The tilde (~) down the left-hand column of the screen indicates that there is no text in the file, not even blank lines. The prompt line (also called status line) at the bottom of the screen echoes the name and status of the file.

5.1.3 Exiting vi

The vi command to exit and save edits is **`ZZ`**. You can also use ex command **`:wq`** to exit and save the edits (**`:wq`** is equivalent to **`ZZ`**). Unlike vi commands the ex commands, introduced by a **`':'`**, require a **`<Return>`** after the command. Without saving you can exit vi with the ex command **`:q!`**.

5.1.4 vi Command Mode

As soon as you enter a file, you are in vi command mode, and the editor is waiting for you to enter a command. Commands enable you to move anywhere in the file, to perform edits, or to enter insert mode to add new text. Commands can also be given to exit the file in order to return to the UNIX prompt.

One of the most used vi commands is **`'i'`**. The i doesn't appear on the screen, but after you press it, whatever you type will appear on the screen and will be entered into the buffer. The cursor mark the current insertion point. To tell i that you want to stop inserting text, press **`<Esc>`**. Pressing **`<Esc>`** moves the cursor back one space and returns vi to vi command mode.

If you have opened a new file and want to insert the words **`this is a new file`** type the keystrokes **`ithis is a new file`**, what appears on the screen is:

```
this is a new file
```

To break a line press **`<Return>`**.

If you don't know whether you are in the vi command mode or the text input mode press **`<Esc>`** once or twice to enter the vi command mode. When you hear a beep, you are in the vi command mode.

5.1.5 ex Command Mode

A **Q** in the vi command mode invokes the ex command mode. At the command line (bottom) the prompt **':'** appears. The command **vi** returns you back to the vi command mode.

In the i command mode you can call one ex command and immediately return to the vi mode by prefacing a ex command with a **':'**.

5.1.6 Basic vi Keystrokes

Moving around

l or →	Move **forward** one character (**right**).
h or ←	Move **backward** one character (**left**).
k or ↑	Move to **previous** line (**up**).
j or ↓	Move to **next** line (**down**).
w	Move word **forward**.
b	Move word **backward**.
0	Move to **begin** of line.
<Return> or +	Move to **begin** of next line.
-	Move to **begin** of last line.
$	Move to **end** of line.
<Ctrl-f>	Move **forward** one screen.
<Ctrl-b>	Move **backward** one screen.
G	Move to **end** of buffer.
:1	Move to **begin** of buffer.
:n	Move to line number n.
<Ctrl-g>	Display current line number.
<Ctrl-l>	Redraw screen.
/pattern	Search forward for pattern.
?pattern	Search backwards for pattern.
n	Repeat last search in same direction.
N	Repeat last search in opposite direction.

Inserting Text

i	Inserting text before cursor.
a	Inserting text after cursor.
I	Inserting text at begin of line.
A	Inserting text at end of line.
o	Open new line for text below cursor.
O	Open new line for text above cursor.

Deleting Text

X	Delete previous character.
x	Delete character under cursor.
dw	Delete the word the cursor is on.
D	Delete form cursor to end of line.
dd	Delete current line.
p	Put deleted text after cursor.
P	Put deleted text before cursor.

Yank [1]

yw	Yank (copy) word.
yy	Yank (copy) current line.
"ayy	Yank (copy) current line into named buffer a.
p	Put yanked text after cursor.
P	Put yanked text before cursor.
"aP	Put text from buffer a before cursor.

Undoing and other vi Commands

.	Repeat last edit command.
u	Undo last edit.
U	Restore current line.
J	Join two lines.

Exiting Commands

ZZ	Save (write) and quit file.
:x	Save (write) and quit file.
:wq	Save (write) and quit file.
:w	Save (write) file.
:w ***filename***	Write current buffer to *filename*.
:q	Quit file.
Q	Quit vi and invoke ex.
:e ***file***	Edit *file* without leaving vi.

Some ex Commands

:set	Display options set by user.
:set all	Display list of all current options, both default and those set by the user.
:set number	Display line numbers.
:set showmode	Displays in insert mode a message on the prompt line indicating the type of insert you are making.
:set ***option***	Activate *option*.
:set ***option=value***	Assign *value* to *option*.
:set ***option*****?**	Display *value* of *option*.
:set no** *option* **	Deactivate *option*.

[1] Yanking means copying text into a buffer

:sh	Invoke shell.
<Ctrl-d>	Return to editor from shell.
!*command*	Execute UNIX *command*.
:r *newfile*	Read contents of *newfile* into current file.
:r !*command*	Read output of UNIX *command* into current file.

5.1.7 The .exrc File

Your own vi environment is controlled by the **.exrc** file in your home directory. A sample .exrc file looks like this:

```
set number
set showmode
```

The file is read by ex before it enters the vi mode, commands in .exrc should not have a preceding colon.

5.1.8 More about vi

More about vi can be found in the Hypertext reader of your UNIX system and in the man page.

Recommendable books about vi are 'Learning the vi Editor' from Linda Lamb [12] and 'The Ultimate guide to the vi and ex text editors' from the Hewlett-Packard Company [5].

5.2 GNU Emacs

GNU Emacs is a powerful editor in the UNIX world. Emacs belongs to the GNU project of the Free Software Foundation (look appendix A) and is available on nearly all UNIX platforms, like AIX, HP-UX and ULTRIX. We have it running on AIX and HP-UX.

Unlike most other editors, Emacs is a complete working environment. You can start Emacs in the morning, work all day and night, and never leave it. You can use it to edit, rename, delete, and copy files; to compile programs; to interactive work with the UNIX shell; and so on. Before windowing systems like X became popular, Emacs often served as a complete windowing environment.

5.2.1 Emacs Commands

Emacs commands consist of a modifier, such as **<Ctrl>** (CONTROL), **<Esc>** (ESCAPE), or **<META>** (META), followed by one or two characters. In this text the following notation is used to describe the keystrokes.

<Ctrl-g>	Hold down the **<Ctrl>** key and press **g**.
<ESC>-x	Press **<Esc>**, release it, and then press **x**.

Most Emacs manuals refer to the **<META>** key instead of the **<Esc>** key. But most keyboards don't have a **<MEAT>** key, so we will refer to **<Esc>**. If you have a **<META>** key, you will probably refer to use it instead of **<Esc>**. The **<META>** key works like the **<Ctrl>** key described above. **<ESC>-x** is then equivalent to:

<META-x>	Hold down the **<META>** key and press **x**.

To complete a command you may need to press a carriage return:

<Return>	Press the **RETURN** key. This key may be labeled **ENTER** on your keyboard.

All Emacs commands, even the simplest ones, have a 'full name': for example **forward-word** is equivalent to the keystrokes **<ESC>-f** and **forward-char** is equivalent to **<Ctrl-f>**. Many commands only have 'full names'; there are no corresponding keystrokes.

5.2.2 Starting Emacs

To start Emacs, simply type emacs followed by the name of the file you want to edit.

emacs *myfile*

5.2.3 Exiting Emacs

To exit emacs, type

<Ctrl-x> <Ctrl-c>

5.2.4 Emacs Screen

When you enter Emacs, you are in a workspace. A cursor marks the position in the file. You don't have to do anything special before you start typing.

Just above the bottom of the screen, Emacs prints information about what it is doing. This line is called the 'mode line' and may look like this:

```
--**-Emacs: myfile (Text Fill)---5%----
```

At the left edge of the mode line, you may see two asterisks (**). This means that whatever you're editing has been modified since the last time you saved it. If you haven't made any changes, the asterisks won't be there. Next, Emacs prints 'Emacs:' followed by the name of the buffer or file you are editing (myfile). In parentheses following this Emacs shows the major (Text mode) and minor mode (Fill mode). Following this Emacs prints where you are in the buffer or file (5%). If the entire file is visible on the screen, Emacs prints the word ALL.

5.2.5 Emacs modes

Emacs has various editing modes in which it behaves slightly differently. When you often want features like word wrap so you don't have to press **<Return>** at the end of the line. When you are programming, your code must be formatted. For writing, there's the **text mode** and for programming in C is the **C mode**.

Text mode and **C mode** are major modes. A buffer can be in only one major mode at a time; to exit a major mode, you have to enter another one.

Whenever you edit a file, Emacs attempts to put you into the correct major mode for what you are going to edit. If you are editing a file with the ending .c, it puts you in the C mode. If the file has the ending .tex, it puts you in the TEX mode. If Emacs can't determine a special mode, it puts you in the fundamental mode, the most general of all modes.

You can also change the mode manual with the command:

<ESC>-x `startup-command` **<Return>**.

The important major modes and there **`startup-commands`** are in the following table:

Mode	Description Startup-command
Fundamental	The default mode; no special behavior. `fundamental-mode`
Text	For writing text. `text-mode`
Directory	For editing directory contents `dired-mode`
Indented text	Indents all the text you type. `indented-text-mode`
Picture	For creating simple drawings. `picture-mode`
C	For writing C programs. `c-mode`
FORTRAN	For writing FORTRAN programs. `fortran-mode`
Emacs LISP	For writing Emacs LISP functions. `emacs-lisp-mode`
LISP	For writing LISP programs. `lisp-mode`
LISP interaction	For writing and evaluating LISP expressions. `lisp-interaction-mode`
nroff	For formatting file for nroff. `nroff-mode`
TEX	For formatting file for TEX. `tex-mode`
LATEX	For formatting file for LATEX. `latex-mode`
Scribe	For formatting file for Scribe. `scribe-mode`
Outline	For writing outlines. `outline-mode`
View	For viewing files but not editing. `view-file`
Mail	For sending mail. `mail`
Read Mail	For reading mail. `rmail`

In addition to the major modes there are also minor modes. These define a practical aspect of Emacs and can be turned on and off within a major mode.

Abbrev	Allows you to use word abbreviations. `abbrev-mode`
Fill	Enable word wrap. `auto-fill-mode`
Overwrite	Replaces characters as you type instead of inserting them. `overwrite-mode`
Auto-save	Saves your file automatically. `auto-save-mode`

In your `.emacs` file, the startup file of Emacs, you can set your favorite modes turned on automatically every time you start Emacs.

5.2.6 Basic Emacs Keystrokes

Moving around

<Ctrl-f>	Move **forward** one character (**right**).
<Ctrl-b>	Move **backward** one character (**left**).
<Ctrl-p>	Move to **previous** line (**up**).
<Ctrl-n>	Move to **next** line (**down**).
<ESC>-f	Move word **forward**.
<ESC>-b	Move word **backward**.
<Ctrl-a>	Move to **begin** of line.
<Ctrl-e>	Move to **end** of line.
<Ctrl-v>	Move **forward** one screen.
<ESC>-v	Move **backward** one screen.
<ESC>->	Move to **end** of buffer.
<ESC>-<	Move to **begin** of buffer.
<Ctrl-l>	Redraw screen with current line in the center.

Deleting Text

<Del> Delete previous character.
<Ctrl-d> Delete character under cursor.
<ESC>-<Del>
Delete previous word.
<ESC>-d Delete the word the cursor is on.
<Ctrl-k> Delete form cursor to end of line.
<Ctrl-w> Delete region (area between mark and cursor).
<ESC>-w Copy region into kill ring.
<Ctrl-y> Restore what you have deleted.
<Ctrl-@> or <Ctrl-<Space>>
Mark the beginning (or end) of a region.
<Ctrl-x> <Ctrl-x>
Exchange point (cursor) and mark.

Stopping and Undoing Commands

<Ctrl-g> Aboard current command.
<Ctrl-u> Undo last edit (can be done repeatedly).

File-handling and exiting

<Ctrl-i> Insert file at cursor position.
<Ctrl-x> <Ctrl-s>
Save file (or may hang terminal; use <Ctrl-q> to restart).
<Ctrl-x> <Ctrl-w>
Write buffer contents to file.
<Ctrl-z> Suspend Emacs (use `exit` or `fg` to restart).
<Ctrl-x> <Ctrl-c>
Exit Emacs.

Tutorial and Getting Help

<Ctrl-h> <Ctrl-h> <Ctrl-h>
Menu of help options.
<Ctrl-h> t Starts Emacs tutorial.

5.2.7 More information about Emacs

There is a man page available with:

`man emacs`

A Postscript file of the reference manual is in the directory `/usr/local/doc/gnu` on the central servers (nearly 300 pages).

A very good book about Emacs is 'Learning GNU Emacs' from D. Cameron and B. Rosenblatt [2]. It covers the first basics and the more advanced features of Emacs.

5.3 Other Editors

A lot of other Editors are available on UNIX systems.

5.3.1 ed Editor

The `ed` editor is a line editor that is available on all UNIX systems and accessible from any terminal. Like the Vi editor, the ed editor has command and input modes, but the ed editor allows you to move and copy only complete lines of data. The ed editor can be used to edit within a shell program or to edit very large files.

For more information about ed see the man page (`man ed`) or the Hypertext reader (`info`) on your UNIX system.

5.3.2 LPEX

LPEX, the Live Parsing Editor, is a programmable editor from the AIX SDE (Software Development Environment) WorkBench/6000 environment. It is based on the OS/2 LPEX editor.

LPEX can be use to create and edit many kinds of data, including programs and documentation.

LPEX provides many capabilities for editing and manipulating documents. You can:

- Use multiple windows to display different documents or more than one view of the same document.
- Select a block of text and move or copy it within or between documents, or to a shell or to another application.
- Search for and change specific text or search for marks in a document.
- Control how often changes are automatically saved and recent changes can be undone.
- Change the appearance of a document by changing its fonts.
- Invoke a parser to use various techniques, such as color and indentation, to display the document.
- Perform many functions using either menus or keys on the keyboard.

LPEX has a menu bar at the top of the application window which contains pull-down menus to initiate commands. Mnemonic keys are also available for command invocation. In addition, LPEX commands can also be entered through the LPEX command dialog invoked from the Command menu pull-down.

LPEX is only available on AIX RS/6000 systems.

5.3.3 INed Editor

The INed editor is a full-screen AIX editor that allows you to do the following:

- Enter subcommands by command keys rather than on a command line.
- Edit multiple files in multiple windows on the screen.
- Scroll the screen horizontally (as well as vertically) through files.
- Move, copy, and delete blocks (as well as lines) of data.

The INed editor also includes:

- A file manager for the following:
  - Creating, changing, deleting, and recovering files and directories.
  - Moving among files and directories.
  - Moving and copying files and directories.
  - Restricting access to files and directories.
- Commands for working with structured files and their histories.
- Commands for limited text formatting.

The INed Editor is available only on AIX systems.

To edit the file *filename* start INed with:

`e` *filename*

More information can be obtained from the man page (`man e`) or the Hypertext reader (`info`).

5.3.4 edt+

edt+ is a screen-oriented text editing utility. This editor emulates the functions and facilities of Digital Equipment Corporation's editor edt. Users who are familiar with Digital Equipment Corporation computers should find that this utility eases the transition between DEC and Unix based systems by providing a common text editing tool. edt+ incorporates a powerful word processor, disaster recovery system, an extensive help facility, full feature programmable text processing and the GOLD-KEY style of editing.

If edt+ is installed on your system, you have to type

`edt` *filename*

to edit the file *filename*.

More information about edt+ in the man page (`man edt`).

5.3.5 uni-XEDIT editor

uni-XEDIT is a UNIX version of IBM's XEDIT editor, part of the VM/CMS mainframe operating system. It provides a very easy to use full-screen editing function. It contains extensive features such as multiple file editing, prefix commands which visually perform line-oriented editing functions, and much more.
This editor is especially useful to users who are already familiar with IBM's mainframe text editors such as XEDIT or SPF EDIT.
We have installed uni-Xedit up to now only on HP-UX.
To invoke uni-Xedit type :
`xe` *filename*

5.3.6 Vuepad

Vuepad is a text editor for use within the X Window System environment. Vuepad allows the use of the mouse for moving the edit cursor and for selecting portions of text for editing operations. Vuepad also supports the VUE drag and drop, allowing the user to drag file icons from the VUE file manager onto the vuepad window for editing.
The Vuepad editor is only available on HP-UX systems.
To start Vuepad type:
`vuepad` *filename*
For more information about Vuepad see the man page (`man vuepad`) or the Hypertext-reader (`info`) on the HP-UX system.

5.3.7 Notepad

The Notepad editor is a self explaining, very useful DEC window text editor. It allows you to use the mouse for moving and for selecting portions of the text.
Notepad is only available on ULTRIX.
To edit the file *filename* enter the command:
`dxnotepad` *filename*
For more information about editing with the Notepad editor double click on the help item inside the Notepad editor.

5.3.8 xedit

xedit is a simple text editor for the X Window System environment. It displays the text of the file in a X Window.
If xedit is installed on your UNIX system, you can edit the file *filename* with the command:
`xedit` *filename*
For more information see the man page of xedit (`man xedit`).

Chapter 6: Text processing

Text processing, word processing, text editing, and other phrases are often used to describe activities dealing with the input, editing and outputting of written forms of expressions. The important part of input of text into a file without further text formatting is described in chapter 5.

In this chapter, the text processing system TEX (and LATEX) will be discussed. Other text processing systems and the Unix-tools `awk` and `sed` to automate the editing, analyzing and processing of text will be described in a later release of this primer.

6.1 TEX and LATEX text processing

TEX [8, 9] (pronounced `Tekh`) is a text processing system developed by Donald E. Knuth of Stanford University to compose by computer high quality documents, especially those containing many mathematical formulae. Its typographic quality is comparable to the finest works of the printing art. The TEX text formatting system offers several tens of custom-made fonts and over 300 basic and 600 "composed" `PLAIN` commands.

Unlike most desktop publishing systems, TEX does not attempt to show the appearance of the finished document on the screen as it is edited (although screen previewers are available). Instead, the document is "marked up" with codes, which indicate boldface, italics, special characters (e.g., "`\int`" for an integral sign), and the like. Large-scale aspects of design are automated; you just give the title of a chapter, and let the computer take care of numbering the chapter and putting the title in the right place on the page.

TEX is generally considered the most sophisticated computer typesetting system, as well as (for an experienced user) one of the easiest to use. LATEX is a macro package build on top of TEX.

Fundamental to LATEX is the idea of a **document style** which determines exactly how a document will be formatted. LATEX provides standard document styles that describe how standard logical elements should be printed. One may have to supplement these styles by specifying the formatting of logical structures peculiar to a given document, such as mathematical formulae. One can modify the standard document styles or create an entirely new one, though one should possess a basic understanding of typographical design before creating a radically new style.

6.1.1 Advantages of LATEX

- The layouts were developed by (American) design professionals, and they produce documents which indeed look as though "they were printed".
- It is extremely simple to compose mathematical formulae.
- The user must only know a few easily memorized commands, which control the logical structure of the document, and (almost) does not have to know the technical details about how a document is formatted.
- More complex components such as footnotes, bibliography, table of contents etc. as well as simple pictures can be produced without great difficulty.
- LATEX is the "de facto" standard for scientific publications in Europe and the U.S.A..

6.1.2 Disadvantages of LATEX

- The document can only be output on display units or printers with graphical (all point addressable) capabilities.
- One can vary a few parameters quite easily within the framework of a given LATEX document layout style. However more basic deviations from the available layouts are only possible with a lot of (re)programming.

6.1.3 How does LATEX work?

A TEX (LATEX) input file is a "plain" text file prepared with a text editor and in general has the extension `tex`.

The files describing the document styles or options (extension `sty`) are stored in a standard public directory (`/usr/local/lib/tex/inputs`). There exist many LATEX styles, but only four standard ones, namely, `report`, `article`, `book` and `letter`. These styles can be further customized by specifying one or more options.

The output from LATEX is a set of files. One of them (extension `dvi`) contains a device independent binary representation of the formatted text. This file can be converted into a printable form by a separate program.

A logfile of the (La)TEX run is generated (extension `log`); it contains information which also appeared on the screen (e.g. the names of the files read, the numbers of the pages processed, error messages etc).

The other files contain information about cross-referencing (extension `aux`), table of contents (extension `toc`), list of figures (extension `lof`) and list of tables (extension `lot`). They are used in a subsequent LATEX run to produce particular elements of the document.

A `idx`-file contains all indexed items. They can be sorted and included as an `ind`-file. To produce this sorted index, the program `makeindex` is provided.

A `bbl`-file contains the bibliographic references extracted by BIBTEX from the bibliography database.

Figure 6.1 summarizes the dependencies of the different files used by LATEX.

6.1.4 TEX glossary

This section is partly based on a glossary posted to `TEXHAX` by Jacques Goldberg of the Department of Physics, Technion–Israel Institute of Technology, Haifa (Israel).

glyph A graphical representation of a character, for example **a**, *a* and `a` are three different glyphs for the same character.

font A set of glyphs corresponding to a character set.

primitives About three hundred basic commands wired into the TEX program.

macros More sophisticated or simpler to use commands, built upon the primitives.

Plain TEX Donald Knuth's TEX augmented by his own set of macros.

LATEX Donald Knuth's TEX augmented by Leslie Lamport's alternative set of macros.

$\mathcal{AMS}$-TEX Donald Knuth's TEX plus American Mathematical Society macros. The functionality of this package is now available to LATEX users with the `amstex` style option.

format A given set of macros, compiled and written into a file of filetype `fmt` once to save repeated compilation time. TEX will restore all definitions by reading such a format file.

TFM Font Metric files contain *device independent global* information about the space on the output requested for each glyph in the font, as well as other boundary conditions such as relative spacing with neighboring glyphs (e.g. ligatures). TEX does *not* need to know the detailed shape of a glyph. Filetype is `tfm`.

DVI A DeVice Independent file (extension `dvi`) is output by TEX. DVI files use the ASCII 255-character set and can be copied between different computers (PC's and mainframes) as *binary (image)* files. They contain the description of the formatted document.

DVI driver A program which turns the human-unreadable DVI file into ink on paper or light on a display. The DVI driver *must* know exactly how to produce the image of a given glyph. Therefore it needs a *raster* image for each glyph, and this raster is device (printer, display) dependent, at least, but not only, because devices have different resolutions.

previewer DVI driver to output the DVI file onto a screen.

font file A set of raster representations for each glyph in a font, clearly printer and magnification-dependent. These files have various filetypes such as `pxl`, `pk`, `gf`. The differences in names are historical and technical. `pxl` is obsolete but still used by some old drivers, `pk` is by far the most efficient especially when it comes to saving disk space, `gf` is produced on the way to making fonts by the METAFONT program.

DVI drivers are *not* supposed to be able to magnify fonts, but they should not generate a fatal error when a font is not available. DVI drivers expect to find the magnified rasters in a specific directory structure, and the user is responsible for not invoking magnifications for which no raster files exist; again TEX does *not* check that the driver will or will not be able to print out a font at the requested magnification. This is *not* TEX's business.

An exception is the `dvips` driver. If it finds a missing font, it tries to calculate the font in the required magnification from the METAFONT input files (extension `mf`). They are then stored in the `/usr/local/Localfonts` directory.

magnification A scale factor applied to the original design size for a font. The following magnification factors are accepted:

$$0.5, 0.6, \ldots, 1.0, \sqrt{(1.2)}, 1.2, (1.2)^2, etc$$

Not all fonts at each magnification will be present on all machines.

METAFONT A companion program to TEX used to generate new fonts, or existing fonts at new magnifications. The program METAFONT reads

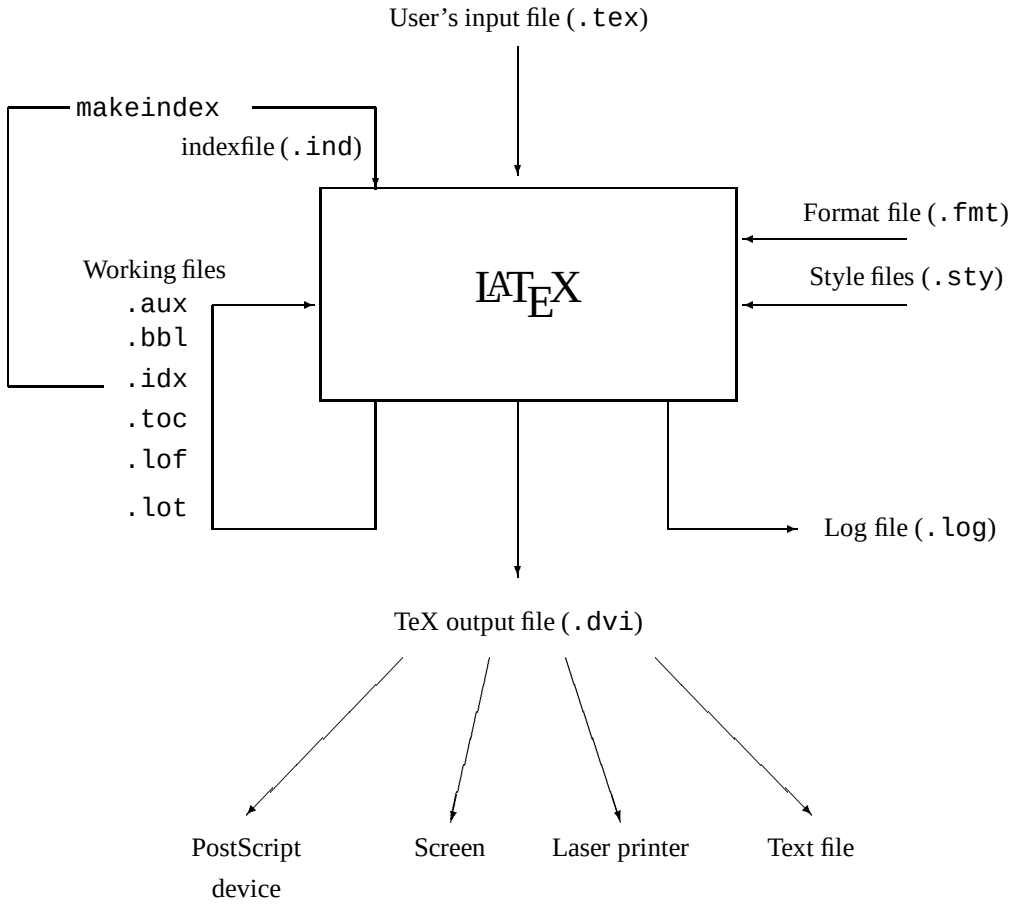


Figure 6.1: Data flow for the files used by LaTeX

a font definition file, actual device parameters, magnification etc., to generate *two* output files per input font file definition. The input is an `mf` file and the two outputs are the corresponding `tfm` and `gf` files. If you wish to make a font at 5 different magnifications you have to run METAFONT 5 times, on almost the same `mf` file. If you wish to create the font for 3 different devices (a dot matrix printer, a laser printer, a display) you again must run METAFONT 3 times, with the correct printer definition.
All these runs will always produce the same `tfm` file but a *different `gf` file.*

LATEX **style** A file which contains the description of or changes to a document layout style (extension `sty`). These are the files which are specified in the LATEX command

```
\documentstyle[minor_style]
    {major_style}
```

where `minor_style` is an optional part, which specifies small changes to be made to the mandatory `major_style`.

6.1.5 Documentation in PostScript form

All documentation related to TEX, LATEX, `sty`s, drivers etc. are located in the `/usr/local/doc/tex` directory. You can `cd` to there and have a look on the various documents and print them with the usual `lp, lpr` commands. Often you will find documentation on `sty` files also in the `/usr/local/lib/tex/inputs` directory.

6.1.6 Suggested Reading

In order to exploit the power of TEX and LATEX you should consult one of the many books available. Below is given a short selection of the literature, as well as some articles in TEX or PostScript form:

- *The TEXbook* by D. Knuth, Addison Wesley [8]
- LATEX, *a Document Preparation System* by L. Lamport, Addison Wesley [11]
- LATEX, *Eine Einführung* by H. Kopka
- *Kompaktführer* LATEX by R. Wonneberger
- LATEX *Kurzbeschreibung* by H. Partl, `/usr/local/doc/tex/lkurz.ps`

6.1.7 Running LATEX

The latest versions of TEX, METAFONT and the LATEX macro package have been installed on all centrally supported HP and IBM UNIX workstations at GSI. Below we show the basic commands to format a LATEX document, preview the document on a graphics screen (X11), transform the `dvi` into PostScript and print the output. Then each major system will is reviewed in turn.

Running a Sample File

Before preparing your own documents, you may want to get acquainted with LATEX by running it on a sample input file. You can copy the file `sample.tex`, which is present in the `/usr/local/lib/tex/inputs` directory.

You can now modify your copy of the file `sample.tex` if you want and then run LATEX on your file by typing:

```
latex sample
```

When LATEX has finished, it will have produced the file `sample.dvi` in your current (working) directory. You can view this file on your X-screen by the command

```
xdvi sample.dvi
```

In general the extension `.dvi` can be omitted.

You can print this file by first transforming the information into (file extension `ps`) PostScript using the command `dvips`.

```
dvips sample.dvi
```

In general the extension `.dvi` can be omitted.

At GSI, dvips is configured so that it produces a file `sample.ps`. You can either view this PostScript file on your X-screen with the command

```
ghostview sample.ps
```

or print it on a PostScript laserprinter:

```
lp sample.ps          -- on HP
lpr sample.ps         -- on AIX,ULTRIX
```

To specify a printer or use other options for that specific printer, see chapter 4.5 on page 23.

After your output has been printed, you can delete `sample.dvi` and possibly `sample.ps`.

Going into more details

The `tex`, `latex` and other relevant commands reside in the `/usr/local/bin` directory. When you get a `latex: not found` error from the system, first check whether the above directory is in your command search path by typing `echo $PATH`. If it is, please type `ls /usr/local/bin/latex` to check whether the `latex` file is indeed installed. If it is not, contact your system administrator.

Once you get LaTeX started, you should get the message:

```
This is TeX, C Version 3.14t3
```

If the command `latex` is found, but you do not get this far, then you have probably the wrong version of the LaTeX format. In this case you should contact your system administrator. Please remind that the LaTeX version installed on the Unix workstations is newer than on the IBM MVS and VAX VMS systems and uses the New Font Selection Scheme (NFSS) developed by Frank Mittelbach and Rainer Schöpf [13]

Interrupting TeX and respond to ? messages

When TeX is running you can interrupt it by typing `<Ctrl-c>`. This will stop TeX as if it had encountered an ordinary error, and you can then return to Unix shell command level by typing `X`, as described in the TeXbook.

If TeX stops with an error message and a question mark (?), you have several possibilities to proceed:

`x`	stop the execution of TeX
`h`	ask TeX for a more detailed information about the error
`_`	(just press `<Enter>`): continue and hope that TeX can go ahead in spite of the error encountered
`e`	call the editor of your `.tex` file and jump to the line in error
`r`	run without stopping
`s`	scroll future error messages
`q`	run quietly
`i`	to insert something, e.g. to correct a misspelled control sequence

A "deadlock" may occur if LaTeX requires a file which is not accessible. The user will be prompted to input the correct file name, if any. If you want to exit from TeX at that point, specify `null` as filename. This is an empty file, which is provided in the standard input file directories.

You should be aware that TeX only searches its own input directories. By defaults, these are the system inputs directory (`/usr/local/lib/tex/inputs`) and the current working directory. You can specify other directories by relative or absolute pathnames. The ~ character for the home directory is not recognized.

`dvips` - DVI converter to PostScript

The DVI driver `dvips` translates a DVI file into a PostScript file. The latter can be printed on any PostScript printer, usually the HP Laserjet IIISi or a DEC PostScript printer.

The syntax for the `dvips` command with a selection of important options is:

```
This is dvips 5.482 Copyright 1986-92
       Radical Eye Software
    Usage: dvips [options] filename[.dvi]
 l # Last page
 n # Maximum number of pages
 o f Output file
 p # First page
 q*  Run quietly
 r*  Reverse order of pages
 c # Uncollated copies
 C # Collated copies
 E*  Try to create EPSF

    # = number   f = file   s = string
    * = suffix, '0' to turn off
    c = comma-separated dimension pair
        (e.g., 3.2in,-32.1cm)
```

Many other options can be found in the `man` pages, in the file `/usr/local/lib/tex/doc/dvips.ps` or by `dvips -?`.

The DVI file may in general be specified without the `.dvi` extension. Fonts used may either be resident in the printer or defined as bitmaps in `pk` files, or a "virtual" combination of both. `dvips` will automatically invoke METAFONT to generate fonts that don't already exist.

If you want to print your file, use the usual printer commands:

```
lp [-d myprinter] sample.ps          -- on HP
lpr [-P myprinter] sample.ps         -- on AIX,
                                        ULTRIX
```

For a description of the other options of the `lp, lpr` command and possible printers see chapter 4.5 on page 23.

You can direct the output from dvips directly to the printer without writing to the intermediate PostScript file:

```
dvips -o!lp sample.dvi      -- on HP
dvips -o!lpr sample.dvi     -- on AIX,ULTRIX
```

`xdvi` – DVI previewer for X-windows

The DVI previewer `xdvi` is a program which runs under the X window system. It has the capability of showing the file shrunken by various (integer) factors, and also has a "magnifying glass" which allows one to see a small part of the unshrunk image momentarily.

In addition to using keystrokes to move within the file, `xdvi` provides buttons on the right side of the window, which are synonymous with various sequences of keystrokes.

```
xdvi [+[<page>]] [-s <shrink>]
[-paper <papertype>] [-mgs[n] <size>]
[-fg <color>] [-bg <color>] [-hl <color>]
[-bd <color>] [-cr <color>] [-bw <width>]
[-geometry <geometry>]
[-iconic] [-display <host:display>]
dvi_file
```

The many other options of `xdvi` are described in the `man` page or by typing `xdvi`.

`gs` and `ghostview` – a PostScript previewer

The public domain Ghostscript previewer `gs` is available on Unix workstations. It is part of the GNU project. Ghostscript is a programming language similar to Adobe Systems' PostScript (tm) language. `gs` reads a file and displays it as a Ghostscript file. It then interprets commands from standard input until an end-of-file character (`<Ctrl-D>`) is typed. The `'quit'` character (`<Ctrl-C>`) also terminates Ghostscript execution.

To invoke the interpreter, give the command

```
gs <filename1> ... <filenameN>
```

The interpreter will read in the files in sequence and execute them. After doing this, it reads further input from the primary input stream (normally the keyboard). Each line (i.e. characters up to a `<return>`) is interpreted separately. To exit from the interpreter, type `quit<return>`. The interpreter also exits gracefully if it encounters end-of-file.

The program `ghostview` provides a menu/mouse interface for the `ghostscript` previewer. To invoke this interface, give the command

```
ghostview  <filename>
```

You can then easily browse through your document, select specific pages, print the whole document or only selected pages, see the document with all PostScript figures included and zoom it.

6.1.8 Using PostScript fonts

Thanks to the NFSS (new font selection scheme) by Schöpf and Mittelbach [13] it is easy to replace the computer modern fonts provided with TEX by the arbitrarily scalable original Adobe PostScript fonts. This task has been simplified by the use of some document-style suboptions to be included in the `\documentstyle` command. The following style options and new commands are available:

`avant` makes AvantGarde default text font, and Times default sans font.

`bookman` makes Bookman default text font, and Helvetica default sans font.

`dingbat` sets up various commands which use the Zapf Dingbats font, each of which take as a parameter the character number of a symbol in the font. Possibilities are:

- `\ding` Just prints the character
- `\dingfill` wherever you use a filler, fills the space with the selected character
- `\dingline` a freestanding line of symbols
- `\dinglist` a list environment which tags items with the selected symbol

`epsfig` a style file for including Encapsulated PostScript figures. See section 6.1.9 below for a description.

`helv` makes Helvetica default text font, and Times default sans font.

`ncs` makes NewCenturySchoolbook default text font, and AvantGarde default sans font.

`palatino` makes Palatino default text font, and GillSans default sans font.

`times` makes Times default text font, and Helvetica default sans font. This document has been produced with this style option.

For a more complete description of the NFSS together withy PostScript fonts, refer to the article `/usr/local/doc/tex/psnfss.dvi`.

Please remind that the dvi-driver has to be able to handle PostScript fonts. At the moment, `xdvi` is *not* able to deal with these fonts. Instead you have to use `dvips` and `ghostview`!

Playing with PostScript

LATEX offers only a certain number of font sizes (. . . , `\Large`, `\LARGE`, `\Huge`). When for a particular

Simple example on how to generate large letters

```
\font\VB=ptmr at 22mm   % declare very big font (Times at 22 mm, K.Berry name 'ptmr')
{\VB Large 22 mm text}
```

Large 22 mm text

Figure 6.2: Large Text with LaTeX and PostScript

purpose a non-available type size or a non-preloaded font is needed, then the basic font commands of TeX can be invoked and the `dvips` driver will then include the font at the desired size. In this case K.Berry's short file name as found in the NFSS font description [13] should be specified. For more details the reader should consult the `dvips` manual, available in the `/usr/local/doc/tex` directory, which describes not only the (short) names under which the various PostScript fonts are known to the TeX programs but it also shows other tricks, which can be used to get special effects with PostScript.

Figure 6.2 shows an example of writing a very large text.

6.1.9 Merging Graphics and Text

Some TeX macro packages allow the creation of graphics e.g. LaTeX's `picture` environment and its extensions, `epic` and `eepic` or `PiCTeX`. These will not be discussed here. We will describe, however, ways to merge text and graphics, if the latter are prepared as Encapsulated PostScript form.

TeX has a primitive command called `\special`, which allows arbitrary text to be included in the `dvi` at the current position. The text given to the `\special` command is **ignored** by TeX itself. This text can however be interpreted by the dvi-driver when it prepares printable or viewable output from the information in the dvi file, in particular it can be used to insert a graphics file. So, the problem of producing proper graphics output is solved if the dvi-driver and the printer can handle a given format.

The EPS format is a set of rules for writing programs in PostScript[1]. An EPS file starts with two characters `%!` followed by any text, and has some metacomments, starting with %%, for instance:

```
%!PS-Adobe-2.0 EPSF-1.2
%%BoundingBox: 40 -505 507 -87
%%Creator:Adobe Illustrator(TM) 1.1
%%Title:elephant.ps
%%CreationDate:25/4/88 7:57 pm
%%DocumentFonts:Courier
%%EndComments
     ...
```

For EPS conformance only the first line and the one containing the `BoundingBox` comment are mandatory. The program `dvips` which converts a DVI file into PostScript, uses the latter metacomment, which specifies the coordinates of the "bounding box" of the picture (see next section).

What is a bounding box?

In order to be able to properly translate and scale a figure, TeX must know its "natural" position on the page; this information is present in what is called the *bounding box* of a PostScript program. The bounding box is an outer limit to the marks created by a program, and is specified as four coordinates of a rectangle: the lower-left x coordinate (bbllx), the lower-left y coordinate (bblly), the upper-right x coordinate (bburx), and the upper-right y coordinate (bbury). Adobe uses the convention that the bounding box of a PostScript program must be contained in a "bounding box comment" if the file is to be used as an Encapsulated PostScript figure. It is a line of the form:

```
%%BoundingBox:  bbllx bblly bburx bbury
```

Note once more that the only mandatory PostScript convention is that the first line of the file should begin with the characters "%!" (a "%" begins a comment in PostScript). A good place for the bounding box comment is as the second line of the file.

All coordinate values must be given in so-called Big Points (72 big points equal one inch, or 1cm = 28.35 big

[1] See "Appendix G: Document Structuring Conventions — Version 3.0" and "Appendix H: Encapsulated PostScript File Format — Version 3.0" in *The PostScript Language Reference Manual* [1]

points), e.g. the bounding box corresponding to an A4 (210 mm by 297 mm) page is:

```
%%BoundingBox: 0 0 595 842
```

If a bounding box comment is present in the figure file, the `epsfig` TEX interface (see section 6.1.9) will extract its values. The bounding box values may instead be specified directly in the `epsfig` argument, using clauses of the form `bbllx=`*`bbllx`*, `bblly=`*`bblly`*, ... in which case the figure file is not searched for the bounding box.

Producing Encapsulated PostScript pictures

Many programs dealing with graphics, produce Encapsulated PostScript files as graphics output files.

With PAW, workstation type 113 will produce PostScript output with the Boundingbox command parameters included. Below the beginning of a PostScript file generated by this procedure with PAW is shown.

```
%!PS-Adobe-2.0 EPSF-2.0
%%BoundingBox: 0 0 567 567
%%Title: /PAW      PS       A1
%%Creator: HIGZ Version 1.13/00
%%CreationDate: 23/01/92   16.19
%%EndComments
/s stroke def /l lineto def
/m moveto def /t  translate def
              ......
```

Other picture formats should first be transformed to PostScript before they can be included easily into LATEX. Several public domain programs exist to transform popular picture formats into PostScript (see section 8.2 on page 48). .

Moreover the utility `XtoPS` dumps an image of an X window as encapsulated Postscript. The image can be color or grayscale.

Usage of the epsfig command

The style option `epsfig`, which is based on the `psfig` macros of T.J. Darrell as extended by S. Rahtz, facilitates the inclusion of PostScript figures into TEX documents. With the help of a compatible DVI postprocessor, PostScript figures are automatically scaled and positioned on the page, and the proper amount of space is reserved. Custom characters such as "⊗" and "◔" may be created and used freely throughout a document, or figures can be presented as traditional broken-out displays (see figs. 6.3, 6.4).

Figure 6.3: Encapsulated PostScript example 1

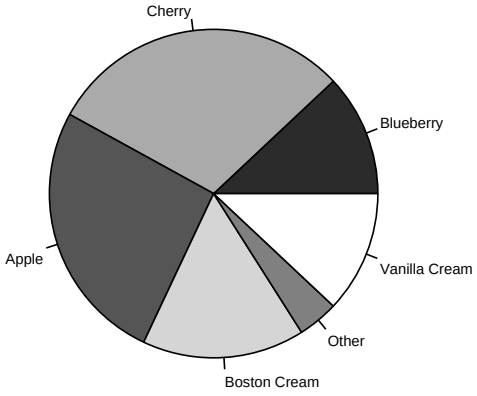


Figure 6.4: Encapsulated PostScript example 2

To include Encapsulated PostScript pictures with `dvips` the style option `epsfig` must be specified as follows:

```
\documentstyle[epsfig,...]{...}
```

The command `\epsfig` is then defined with the following parameters (given in one line !):

```
\epsfig{file=fn,height=ht,width=wd,
bbllx=blx,bblly=bly,bburx=brx,bbury=bry}
```

`file` The file name of the Encapsulated PostScript file.

`figure` Alias for `file=`, i.e. the file name of the Encapsulated PostScript figure.

`height` The desired height of the picture (in any of the accepted TEX units). If this parameter is not specified, then the picture will be printed with its "natural" height, i.e. the one specified on the BoundingBox line inside the PostScript file. When a width is specified (see below) and no height, the latter is scaled in the same proportion as the width.

`width` The desired width of the picture (in any of the accepted TEX units). If this parameter is

not specified, the picture will be printed with its "natural" width, i.e. the one specified on the BoundingBox line inside the PostScript file. When a height is specified (see above) and no width, the latter is scaled in the same proportion as the height.

`bbllx` The **x**-coordinate of the **lower left** hand corner of the BoundingBox of the PostScript picture.

`bblly` The **y**-coordinate of the **lower left** hand corner of the BoundingBox of the PostScript picture.

`bburx` The **x**-coordinate of the **upper right** hand corner of the BoundingBox of the PostScript picture.

`bbury` The **y**-coordinate of the **upper right** hand corner of the BoundingBox of the PostScript picture.

When the BoundingBox parameters are specified on the `epsfig` command, then the ones inside the PostScript file are ignored. This facility is particularly useful if the BoundingBox parameters are absent from the PostScript file or are wrong.

The `\epsfig` macro is (unfortunately) sensitive to whitespace, and will be confused by any extra spaces or newlines in its argument!

Simple figures

The code below shows the simplest way of how one can include an Encapsulated PostScript file with `\epsfig`.

```
\documentstyle[epsfig]{article}
\begin{document}
     .... some  text ....
\epsfig{file=input-file}
     .... some more text ....
\end{document}
```

Here `input-file` is the name of a PostScript file. `epsfig` will automatically position the figure at the current point on the page, and reserve the proper amount of space in TEX so as to avoid blocking any other objects on the page.

As a more realistic example let us include a EPS picture (e.g. generated with PAW) and specify the desired width on the output page (if we do not specify any dimensions, the "natural" dimensions of the picture are taken, as read on the `BoundingBox` line in the file, corresponding to the size shown when the picture is printed separately on a PostScript printer). The upper edge of the picture will be located at the point where the command `epsfig` is issued. The graphic will be scaled to the desired width (or height), but it will be scaled by the same factor horizontally and vertically (if only one of the parameters `height` or `width` are specified). The actual commands typed are given below (the `epsfig` macro must be written in one line!).

```
\begin{figure}
\begin{center}
\mbox{\epsfig{file=tac2dim.eps,
      width=\linewidth}}
\end{center}
\caption{A single centered figure}
\label{fig:simple}
\end{figure}
```

Figure 6.5 shows the resulting picture, with width of the picture equal to the current linewidth. The width (or height) can be given also in cm or any other valid TEX length unit. It is centered by putting it in an `mbox`, which is itself in a `center` environment.

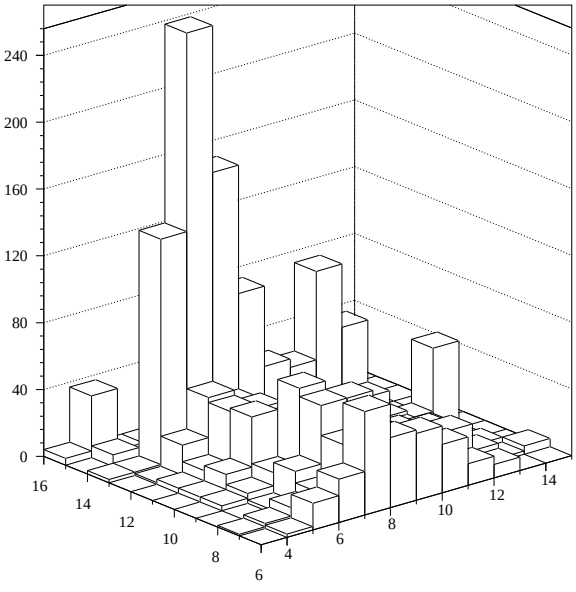


Figure 6.5: A single centered figure

Draft figures and Silent mode

Normally, `epsfig` will print advisory messages to remind you that it is including figures as TeX processes a

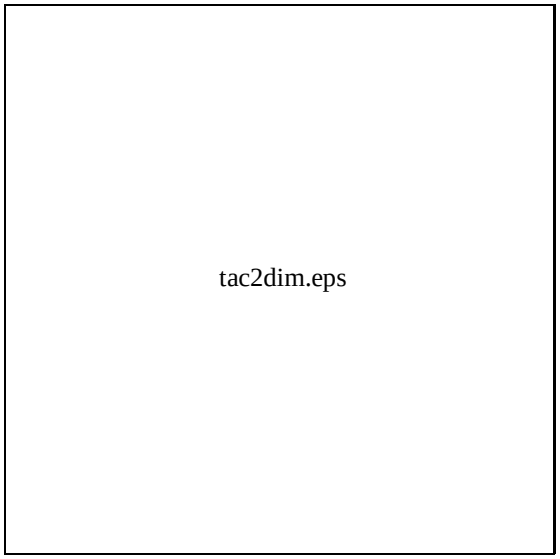


Figure 6.6: The same figure as 6.5, but in draft mode

```
\begin{center}
\mbox{%
\epsfig{file=rosette.eps,
     ,height=20mm,width=6mm}
\epsfig{file=rosette.eps,
      height=20mm}
\epsfig{file=rosette.eps,
      height=20mm,width=35mm}
}% end of \mbox
\end{center}
```

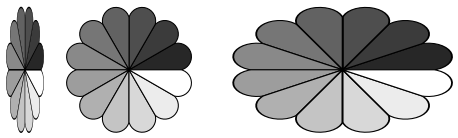


Figure 6.7: Distorting a picture with (`epsfig`)

document. This behavior can be disabled with `\pssilent` and re-enabled with `\psnoisy`.

Some PostScript figures can take quite a long time to transmit to the printer and print; for these figures a "draft" mode is available to speed printing of draft versions of the document. A figure printed in draft mode will appear as a box with the name of the figure file (Figure 6.6). The macro `\psdraft` will switch into draft mode, and all subsequent `epsfig` macros will produce draft figures until reaching the macro `\psfull`, which switches out of draft mode. No `\special` commands are used in draft mode, so a draft document can be previewed using any DVI driver.

Distorting a figure Users who want to obtain special effects by distorting a figure can specify *both* parameters to the `\epsfig` macro, where both dimensions `height` and `width` are taken literally, thus making disproportionate scaling possible. (see figure 6.7).

Colourfull TEX

With the help of the PostScriptdriver `dvips`, it is easy to produce colorfull slides and include colored pictures into your text. To use the color option, you have to include the document substyle option `dpscolor` into your LATEX document or input `dpscolor.tex` into your TEX document.

After having processed your document in the usual way (`latex, dvips`) you can view it with the `ghostview` previewer or print it on the **psteka** color printer in the I/O room (see section 4.5 on page 23 for details and a list of actual printers).

A set of macros is available to use colors for the background and foreground text:

The first macro lets the user specify the background color for the document. It sets the background color for the current page and all succeeding pages, unless changed by another command of this type. To change the background color back to the default, issue

```
\background{White}
```

There are two types of text color commands. The first is in the form `\ColorName` (note the uppercase for the color name). It is called a local color command since it takes one argument enclosed in brackets. It writes the contents of its argument in the selected color. This should be used for local or nested color changes, since it restores the original color state when it completes. The second type of color command is in the form `\textColorName`. This uses the same naming convention as before. It is called a global color command since it takes no arguments and simply sets the color at this point.

As an example, to write this peace of text in JungleGreen, use the following command:

```
\JungleGreen{write this peace of
text in JungleGreen}
```

In table 6.1 the list of predefined colors is shown. If you want to define your own colors, look into the style file `/usr/local/lib/tex/inputs/dps-color.sty`.

\GreenYellow	\Yellow
\Goldenrod	\Dandelion
\Apricot	\Peach
\Melon	\YellowOrange
\Orange	\BurntOrange
\Bittersweet	\RedOrange
\Mahogany	\Maroon
\BrickRed	\Red
\OrangeRed	\RubineRed
\WildStrawberry	\Salmon
\CarnationPink	\Magenta
\VioletRed	\Rhodamine
\Mulberry	\RedViolet
\Fuchsia	\Lavender
\Thistle	\Orchid
\DarkOrchid	\Purple
\Plum	\Violet
\RoyalPurple	\BlueViolet
\Periwinkle	\CadetBlue
\CornflowerBlue	\MidnightBlue
\NavyBlue	\RoyalBlue
\Blue	\Cerulean
\Cyan	\ProcessBlue
\SkyBlue	\Turquoise
\TealBlue	\Aquamarine
\BlueGreen	\Emerald
\JungleGreen	\SeaGreen
\Green	\ForestGreen
\PineGreen	\LimeGreen
\YellowGreen	\SpringGreen
\OliveGreen	\RawSienna
\Sepia	\Brown
\Tan	\Gray
\Black	\White

Table 6.1: List of predefined colors for use with `dps-color`

Chapter 7: Program Development

7.1 Overview

This chapter introduces program development tools under UNIX with simple examples. Its focus is on commands and command options which work on most UNIX systems at GSI, rather than on the differences.

7.2 Compiling and linking a program

To create an executable program, you compile a source file containing a main program. For example, to compile a C program named `hello.c` use:

```
$ cc hello.c
```

If no errors occur, the compiler creates an executable file named `a.out` in the current working directory. This process is essentially the same for most UNIX compilers. For instance, to compile and run a FORTRAN program use:

```
$ f77 hello.f
$ a.out
```

If your source is divided among separate files, simply specify all files in the compile command:

```
$ cc main.c func1.c ... funcn.c
```

The `-o` *name* option causes the compiler to name the output file *name* instead of `a.out`. For example, to compile a C program `hello.c` and name the resulting executable `hello` use:

```
$ cc -o hello hello.c
```

The `-c` option suppresses the link-edit phase. The compiler generates an object file with the extension (`.o`) for each input file and not the `a.out` file. This is useful when compiling source files that contain only subprograms, which can be linked later with other object files. The resulting object files can then be specified on the compiler command line:

```
$ cc -c func.c
$ cc main.c func.o
```

Notice that you need not call the linker. This is done by the compiler. If you have a lot of different object files you can create an **archive** library to store them. The command `ar` is used to create and manage archive libraries. Its syntax is:

`ar` *keys archive [obj_files]*

The most important *keys* are `r` to replace or add modules to the archive and `t` to display a table of contents. The *archive* is a name composed of `lib`*name*`.a`. For example the command sequence:

```
$ cc -c func1.c
$ cc -c func2.c
$ cc -c func3.c
$ ar r libmy.a func1.o func2.o func3.o
```

compiles `func1.c`,`func2.c`,`func3.c` and adds the object files into the archive `libmy.a`. Libraries are specified on the compile commands with the `-l`*name* option, where *name* is the part of the library name following `lib`. By default the compiler searches the `/lib` and the `/usr/lib` directory for libraries. It is possible to specify additional directories to the search path with the `-L` *libpath* option. For example:

```
$ cc -lmy -L/u/peter/lib main.c
```

looks in the directories `/u/peter/lib`, `/lib` and `/usr/local/lib` in that order for `libmy.a`. There are three more important general available compile options:

`-I includepath`	path for user includes
`-O`	optimize flag
`-g`	debug flag

All options described until now work at least with the following compilers:

- **C**
  - `cc` all flavors
- **FORTRAN**
  - `f77` DEC, IBM RS/6000
  - `fort77` HP
  - `xlf` IBM RS/6000

PL/I compilers are currently available with AIX and Ultrix. Some details will be described in a future release of this document.

C++ compilers are also available. Details will be described in a future release of this document.

7.3 Correcting errors in a program

If your program does not execute properly, you have to use a debugger to locate and correct problems. The old fashioned UNIX debugger `adb` is available on most systems, but it is highly recommended to use the symbolic debugger of your UNIX flavor.

DEC `dxdb`
HP `xdb`
IBM `dbx`
IBM `xde` (X11 interface to `dbx`)

Before invoking a symbolic debugger you should recompile your program with the `-g` option and without any optimization `-O` flags. This ensures that necessary debugging information is incorporated into the object code. The debuggers have many commands for viewing and manipulating programs. You can:

- Control execution with single step execution or the use of breakpoints.
- Look at data values.
- Look at the contents of your source files.
- Look at the execution stack.

A sample of simple commands for the HP `xdb` debugger are:

Command	**Description**
`r`	Run the program
`b 82`	Set a breakpoint at line 82
`c`	Continue running until the next breakpoint
`s`	Single step through the next source line
`S`	Step over a function or subroutine
`t`	Print a trace of the current execution stack
`v`	View a window of lines
`/`*string*	Search forward in the source for occurence of *string*
`?`*string*	Search backward in the source for occurence of *string*
`p abc`	Print the value of variable `abc`
`p abc = 2.2`	Assign a new value to `abc`
`p buffer\ 10d`	Print the first 10 elements of array buffer in decimal format
`D` "*source dir*"	Add directory *source dir* to the list of directories to search for sources (default: current directory)
`q`	Quit the debugger

On DEC and IBM workstations you see the available options via the X11 interface.

7.4 Building and Maintaining a Program

Under UNIX facilities are provided to help to control changes and build a program from many source modules:

- The `make` command builds a program from source modules. Since the `make` command compiles only those modules that were changed since the last build, its use can reduce compilation time when many source modules must be processed (see below and reference [15]).
- The Source Code Control System (SCCS) is a set of UNIX commands that enable you to maintain separate versions of a program without storing complete copies of each version. Beside the reduction of storage requirements the use of SCCS can help in tracking the development of a project that requires keeping many versions of large programs. For more information consult the manuals.

There are two **GNU** tools which can be used instead of the SCCS: the Revision Control System (RCS) and the Concurrent Version System (CVS). If you develop programs in CERN software environment you should consider `cmz` (see section 8.3.2 on page 53) as code management system.

7.4.1 make

The basic operation of `make` is to update a target file. By ensuring that all of the files on which it depends exist and are up to date, `make` creates the target if it has not been modified since its dependents were. It uses information from a description file named `makefile` or `Makefile`, last-modified times from the file system and some built-in rules.

To illustrate, consider a simple example: A program named `prog` is made by compiling and linking three files `x.c`, `y.c` and `z.c` with the library `libS.a`. The files `x.c` and `y.c` share some declarations `xy.h` (that is they have the line: `#include "xy.h"`). The following `makefile` describes these dependencies:

```
prog : x.o y.o z.o
       cc x.o y.o z.o -lS -o prog

x.o  : x.c xy.h
       cc -c x.c

y.o  : y.c xy.h
       cc -c y.c
z.o  : z.c
       cc -c z.c
```

The first line says that `prog` depends on three object files. Once these object files are current, the second line describes how to link them to create `prog`. It is important that such an command line starts with a tab (`<Tab>`) sign. If `x.o` is not up to date the third line says that it depends on `x.c` and `xy.h` and so on. If none of the source or object files had changed since the last time `prog` was made, all of the files would be current, and the command `make` would just announce this fact and stop. If, however, the `xy.h` file had been edited, `x.c` and `y.c` would be recompiled, and then the `prog` would be created.

If no target name is given, the first target mentioned in the `makefile` is created. Otherwise the specified targets are made:

```
$ make x.o
```

would recompile `x.o` if `x.c` or `xy.h` had changed. `make` has a simple macro mechanism. Macros are defined by lines with embedded equal signs. A macro is invoked by preceding the name by a dollar sign. Macro names longer than one character must be parenthesized.

```
CC = cc
prog : x.o y.o z.o
       $(CC) x.o y.o z.o -lS -o prog
x.o  : x.c xy.h
       $(CC) -c x.c
y.o  : y.c xy.h
       $(cc) -c y.c
z.o  : z.c
       $(CC) -c z.c
```

In this example the first line defines the macro `$(CC)` to be `cc`, what is used in the compile steps. If you want to use another compiler you have two possibilities: You can edit that line, or macro definitions on the command line override definitions in the `makefile`. The command `make CC=gcc` uses the GNU C compiler instead of `cc`.

To conclude the structure of an description file:

- `make` ignores blank lines,
- characters from a number (#) sign to the end of a line are comments,
- lines containing an equal (=) sign define macros,
- lines containing a colon (:) are dependency lines,
- lines beginning with a tab (`<Tab>`) sign are command lines.

The most important flag of the `make` command is: `-n` no execute mode (print commands, but do not execute them).

`make` is most useful for medium-sized programming projects, so the typical cycle of program development operations becomes:

think → edit → make → test ...

Chapter 8: Applications and Utilities

8.1 Mathematical packages

There are two interactive systems to meet your needs in scientific computations from symbolics to numerics and graphics:

- **Mathematica** two licenses on rzhp9a,
- **AXIOM** one license on rzri6f.

You can use both systems as:

- A numerical and symbolic calculator.
- A visualization system for functions and data.
- A programming language.
- A modeling and data analysis environment.

Mathematica is widely used. There are user contributed packages from all fields of science. AXIOM is the newest of all symbolic packages. The advantages of AXIOM are:

- The HyperDoc system, offering on-line help, examples, tutorials and reference material.
- A variety of data structures not available in other systems.

A reference manual for each system [16] and [7] is available at the help desk (Benutzerberatung).

8.1.1 Mathematica

To start Mathematica simply type `math` on the rzhp9a:

```
$ math
Mathematica 2.1 for HP 9000 RISC
Copyright 1988-92 Wolfram Research, Inc.
 -- Motif graphics initialized --
In[1]:= Plot3D[Sin[x y],{x,0,Pi},{y,0,Pi}]
Out[1]= -SurfaceGraphics-
In[2]:= PSPrint[%]
Out[2]= -SurfaceGraphics-
In[3]:= Display["!psfix > test.ps",%]
Out[3]= -SurfaceGraphics-
In[3]:= Quit
$
```

Figure 8.1 is the result of the `Plot3D` command. In this example you see two possibilities to plot out of Mathematica. The `PSPrint` command sends a plot to the default printer (psaa). The `Display` command creates a PostScript file.

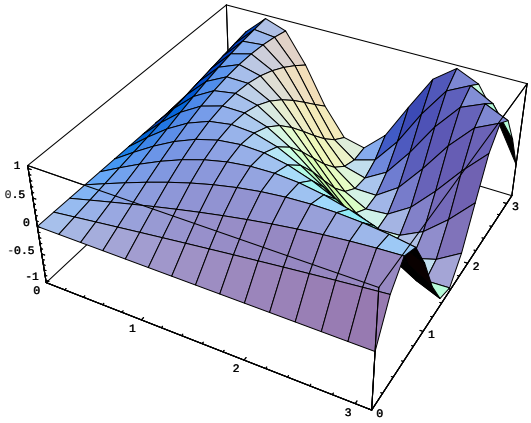


Figure 8.1: Mathematica display

8.1.2 AXIOM

To start AXIOM on rzri6f you have to add `$AXIOM/bin` to your PATH and to type `axiom`. In figure 8.2 you see the top level of the AXIOM hyperdocumentation. It is easy to explore the features of AXIOM by clicking on the items in bold font.

8.2 Graphical Tools

Graphics is used as an essential link between human and computer. Visualization of data allows to extract information much better rather than interpreting raw numbers. Consequently there is a growing demand for high performance graphical tools.

To allow for a free interchange of graphical information between different UNIX-platforms, we support standardized graphical tools and a set of conversion routines to transform from and to different formats of graphical output, which are introduced in the following.

8.2.1 PHIGS

PHIGS (Programmer's Hierarchical Interactive Graphics System) is a programming interface (subroutine package) used in the development of graphics applications. It is based on the American National Standards Institute (ANSI) and International Standard Organization (ISO) standard:

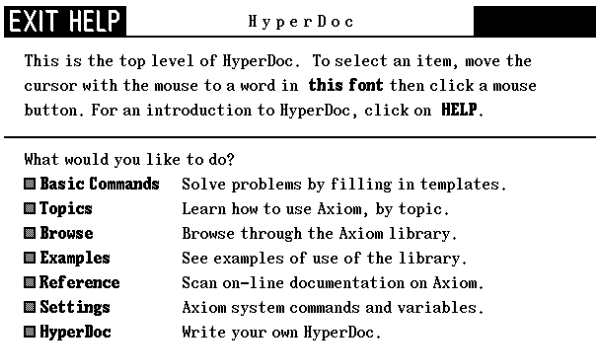


Figure 8.2: AXIOM hyperdoc entry panel

PHIGS is a graphics system that supports the definition, modification, and display of hierarchically organized graphics data. It provides graphics application developers with a significant amount of additional function beyond the CORE and GKS-2D and GKS-3D systems. The system adds new concepts to provide a interactive, three-dimensional system that enhances the design and visualization process. The ability to organize graphic primitives into hierarchical structures makes it possible to edit, modify, and transform graphic entities.

Using over 300 high-level graphic functions, programmers can develop applications in various programming languages. ISO-standard is defined for C and FORTRAN77 binding. On the other hand, a subset of about 50 functions suffices in most application problems.

PHIGS offers a set of device-independent programming tools, it decides whether to use local device processing or to have your Central Processing Unit do the processing for your less intelligent workstations. Identical PHIGS functions are available on both mainframe and standalone environments.

Following is a brief summary of common terms and their definitions:

Primitives Graphic objects are defined by a sequence of elements, including output primitives, attributes, and transformations. Basic output primitive elements include lines, markers, polygons, and text definitions.

Attributes Attributes define the characteristics of an output primitive. An attribute, for example, may define the color of a polyline primitive or size of a polymarker primitive.

Structures The graphical primitives, attributes and model transformations are grouped together to form structures. A structure may be used to represent the geometry of an object as well as information regarding the appearance of that object. Elements may be inserted into, or deleted from, structures at any time, in an operation called structure editing. This editing capability minimizes the need to redefine data in order to modify it. Structures may be related in a number of ways including geometrically, hierarchically, or characteristically, according to your application needs.

Input PHIGS provides essential tools for application interaction. Input devices operate synchronously or asynchronously, relay information to the application, which in turn responds by defining, editing, or displaying the graphical data. PHIGS supports six classes of input devices. These classes represent generic physical devices that differ from one another by the type of data they return to the application. Input device classes include the following:

Locator, Stroke, Valuator, Choice, Pick, String.

Three modes of interaction that allow you to request and obtain data from a logical input device. In REQUEST mode, your application prompts for input and then waits until the operator either enters the requested input or performs a break action which terminates interaction. In SAMPLE mode, your application obtains the current values of the input device by explicitly sampling it. In the EVENT mode, an asynchronous environment is established between your application and a chosen device. In this mode, both your application and any corresponding device operate independently of each other with the help of a centralized input queue.

Workstations The term “workstation” refers to an abstraction of a physical graphics device. It provides the logical interface through which your application program controls physical devices.

PHIGS provides an environment that supports multiple workstations. How your application interacts with a particular workstation depends on the interactive capabilities of that workstation and the

design of your application.

PHIGS supports three categories of workstations: INPUT, OUTPUT, and OUTIN. The capabilities of a workstation determine its category. For example, a workstation such as a plotter may only be capable of generating output. Still another, such as an interactive design station, may be capable of providing both input and output.

Inquiry Functions Inquiry functions allow the application programmer to access the program data contained in state lists, description tables, or structures. They are useful for determining both error conditions and device characteristics.

States The system state defines whether the graPHIGS API has been activated or deactivated, using the Open graPHIGS or Close graPHIGS functions respectively. No other functions can be accessed until the system is "open".

The workstation state defines whether a workstation has been activated or deactivated, using the Open Workstation or Close Workstation functions respectively. The PHIGS structure display functions can only be used if workstation is "open".

The structure state defines whether PHIGS display structure is "open" and able to be modified or closed and unavailable for modification. A structure is opened and closed with the Open Structure and Close Structure subroutines. Graphics primitives and attributes can only be created if the structure state is "open".

Implementations of PHIGS

I. AIX

ISO-PHIGS subroutines are available as a subset of the graPHIGS-IBM library version 2.2 which is available under AIX on RISC6000 workstations and may be used within C-, FORTRAN- and PLI-applications.

Example programs for the use of graPHIGS subroutines are installed in individual sub-directories in the directory:

```
/usr/lpp/graPHIGS/
samples/gettingstarted
```

An interactive tutorial is also available and may be run by just typing: `gPtutor`

```
cc -o square -lgP square.c
```

compiles the file square.c and links with the graPHIGS library yielding the executable square.

The `GDF` file standards are different for AIX and MVS. The command

`cvtgdf`

transforms AIX standard to MVS standard. This converted file may be transferred via FTP to your MVS userid and may be interpreted using TSO GDFIP.

For further information you can use the info-explorer accessing the RAM-DISCS or use the graPHIGS IBM subroutine reference version 2.2 directly.

Literature

IBM-graPHIGS

The graPHIGS Programming Interface,

- Subroutine Reference, version 2.2, SC33-8194-1
- Understanding Concepts, version 2.2, SC33-8191-1
- Writing Applications, version 2.2, SC33-8192-1
- Messages and Codes, version 2.2, SC33-8196-1
- ISO PHIGS Implementation Reference,version 2.2, SC33-8118-00

Graphics in General

- *A Practical Introduction to PHIGS and PHIGS PLUS* by T.L.J. Howard, W.T. Hewitt, R.J. Hubbold, K.M. Wyrwas [6]
- *Computer Graphics: Principles and Practice* by J.D. Foley, A. van Dam, S.K. Feiner, J.F. Hughes [3]

8.2.2 Handling of Image Files

In order to support modification of pixel data and conversion from and to different image file standards the following tools are made available.

University of Utah Raster Toolkit, URT

The Utah Raster Toolkit is a collection of programs and C routines for dealing with raster images commonly encountered in computer graphics. A device and system independent image format stores images and information

about them. Called the RLE format, it uses run length encoding to reduce storage space for most images.

The programs (tools) currently included in the toolkit are listed below, together with a short description of each one. Most of the tools read one or more input RLE files and produce an output RLE file. Some generate RLE files from other information, and some read RLE files and produce output of a different form. For general information on available tools use

man *urt*

For specific information on a chosen command and parameter list type e.g.

man *applymap*

urt - overview of the Utah Raster Toolkit

applymap Apply color map to image data.

avg4 Simple 2x2 downsizing filter.

crop Crop image.

fant Image scale/rotate with anti-aliasing.

getx11 Display using X11.

giftorle Convert gif files to RLE.

graytorle Convert separate rgb files to RLE.

mcut Median cut color quantization.

mergechan Merge colors from multiple images.

pyrmask Generate “pyramid” filter mask.

rawtorle Convert various raw formats to RLE.

repos Reposition an image.

rleClock Draws a clock face.

rleaddcom Add comments to an RLE file.

rleaddeof Add an EOF code to an RLE file.

rlebg Generate a “background”.

rlebox Find bounding box of an image.

rlecomp Image composition.

rledither dither image to a given colormap.

rleflip Flip an image or rotate it 90.

rlehdr Print info about an RLE file.

rlehisto Make a histogram of an image.

rleldmap Load a new colormap into a file.

rlemandl Make a Mandelbrot image.

rlenoise Add noise to an image.

rlepatch Patch smaller images on a big one.

rleprint Print all pixel values in image.

rlequant Variance based color quantization.

rlescale Generate a “gray scale”.

rleselect Select images from an RLE file.

rlesetbg Set the background color of an image file.

rlespiff Simple contrast enhancement.

rlesplice Splice two images horizontally or vertically.

rlesplit Split concatenated images into files.

rleswap Swap or select color channels.

rletoascii Make a line-printer version of an RLE image.

rletogif Convert RLE images to gif format.

rletogray Convert RLE to separate rrr ggg bbb files.

rletops Convert RLE to (BW) PostScript.

rletoraw Convert RLE to rgbrgb raw format.

rletotiff Convert RLE to tiff 24 bit format.

rlezoom Scale image by sub- or super-sampling.

smush Generic filtering.

to8 24 to 8 bit ordered dither color conversion.

tobw Color -> BW conversion.

unexp Convert “exp” format to normal colors.

unslice Paste together “slices” into a full image.

An input file is almost always specified by mentioning its name on the command line. Some commands, usually those which take an indefinite number of non-file arguments (e.g., **rleaddcom**) require a -i flag to introduce the input file name. If the input file name is absent the tool will usually read from the standard input. An input file name of “-” also signals that the input should be taken from the standard input.

On Unix systems, there are two other specially treated file name forms. A file name starting with the character "|"will be passed to sh to run as a command. The output from the command will be read by the tool. A file whose name ends in ".Z" (and which does not begin with a "|") will be decompressed by the `compress` program. Both of these options supply input to the tool through a pipe. Consequently, certain programs (those that must read their input twice) cannot take advantage of these features. This is noted in the manual pages for the affected commands.

An output file is almost always specified using the option `-o` *outfile* . If the option is missing, or if *outfile* is "-", then the output will be written to the standard output.

On Unix systems, the special file name forms above may also be used for output files. File names starting with '"' are taken as a command to which the tool output will be sent. If the file name ends in ".Z", then `compress` will be used to produce a compressed output file.

Several images may be concatenated together into a single file, and most of the tools will properly process all the images. Those that will not are noted in their respective man pages.

Picture comments

Images stored in RLE form may have attached comments. There are some comments that are interpreted, created or manipulated by certain of the tools. In the list below, a word enclosed in <> is a place-holder for a value. The <> do not appear in the actual comment.

`image_gamma=<float number>`

Images are sometimes computed with a particular "gamma" value – that is, the pixel values in the image are related to the actual intensity by a power law,

$$pixel\_value = intensity^{image\_gamma}.$$

Some of the display programs, and the `buildmap` function will look for this comment and automatically build a "compensation table" to transform the pixel values back to true intensity values.

`display_gamma=<float number>`

The display_gamma is just image_gamma. That is, it is the "gamma" of the display for which the image was computed. If an image_gamma comment is not present, but a display_gamma is, the displayed image will be gamma corrected as above. The to8 program produces a display_gamma comment.

`colormap_length=<integer>`

The length of the colormap stored in the RLE header must be a power of two. However, the number of useful entries in the colormap may be smaller than this. This comment can be used to tell some of the display programs (getx11, in particular) how many of the colormap entries are used. The assumption is that entries 0 colormap_length1 are used. This comment is produced by mcut, rlequant, and rledither.

`image_title=<string>`

This comment is used by getx11 to set the window title. If present, the comment is used instead of the file name. (No other programs currently pay attention to this comment.) The comments IMAGE_TITLE, title, and TITLE are also recognized, in that order. No programs produce this comment.

`HISTORY=<string>`

All toolkit programs (with the exception of rleaddcom) create or add to a HISTORY comment. Each tool appends a line to this comment that contains its command line arguments and the time it was run. Thus, the image contains a history of all the things that were done to it. No programs interpret this comment.

`exponential_data`

This comment should be present in a file stored in "exponential" form. See unexp and float_to_exp for more information. The unexp program expects to see this comment.

8.2.3 Image Format Conversion

To convert different image formats, besides URT the command `imconv` (AIX only) supports the following formats:

`gif`, `hdf`, `icon`, `iff`, `mpnt`, `pbm`, `pcx`, `pgm`, `pic`, `pict`, `pix`, `pnm`, `ppm`, `ps`, `ras`, `rgb`, `rla`, `rle`, `rpbm`, `rpgm`, `rpnm`, `rppm`, `synu`, `tif`, `x`, `xbm`, `xwd`.

image conversion is performed due to suffixes, for example,

`imconv` *infile.gif outfile.ps*

converts the gif-format file *infile.gif* to PostScript-format file *outfile.ps* or alternatively

`imconv -xwd` *a* `-ps` *b*

transforms the inputfile named *a* from xwd format to the output file named *b* in PostScript-format.

Under X-Windows, window images may be stored in a dump file. The target window is selected by mouse click. The keyboard bell is ringing once at the beginning of the dump and twice at the end, e.g, the commands:

`xwd -out` *mywindow.xwd*

`imconv` *mywindow.xwd mywindow.rle*

`getx11` *mywindow.rle*

generate a X-window dump file named *mywindow.xwd*, convert it to the RLE-format file *mywindow.rle* and viewed once again.

A truncated subset of the functionality of the commands `xwd` and `imconv` is available under HP-Unix via `XtoPS`. e.g. the command

`XtoPS +border` *mywindow.ps*

generates a postscript-file named *mywindow.ps* excluding the image-borders of the window chosen.

8.3 CERN Software

The CERN program library is a large collection of general-purpose programs maintained and offered in both source and object code from the CERN computer center. Most of the software was developed at CERN and is therefore oriented towards the needs of a physics research laboratory. Nearly all, however, are of a general mathematical or data-handling nature, applicable to a wide range of problems.

The library contains about 3000 subroutines and complete programs. 80% of the programs are written in Fortran77 and the remainder in C or assembly code.

At GSI computing center, the library is available on all supported Unix platforms, i.e IBM AIX, HP HP-UX and DEC ULTRIX, as well as on VMS and MVS. Documentation is available in printed form or as postscript files in the directory `rzhp9a:/cern/doc`.

In the following, some major programs and packages (`paw`, `cmz`, GEANT) are shortly described. Furthermore, an overview of the organization of the CERN-library as well as instructions on how to link own programs with the library are given.

8.3.1 `paw`

`paw` is an interactive utility for visualizing experimental data on a computer graphics display. It may be run in batch mode if desired for very large data analyses. `paw` combines a handful of CERN Program Library packages that may also be used individually in users applications dealing with experimental data. Figure 8.3 shows the various components of `paw`.

The following list points to some typical `paw` applications (for more details, the PAW-manual should be consulted):

- Plot a vector of data fields for a list of events (Ntuples)
- Histogram of a vector of variables for a list of events

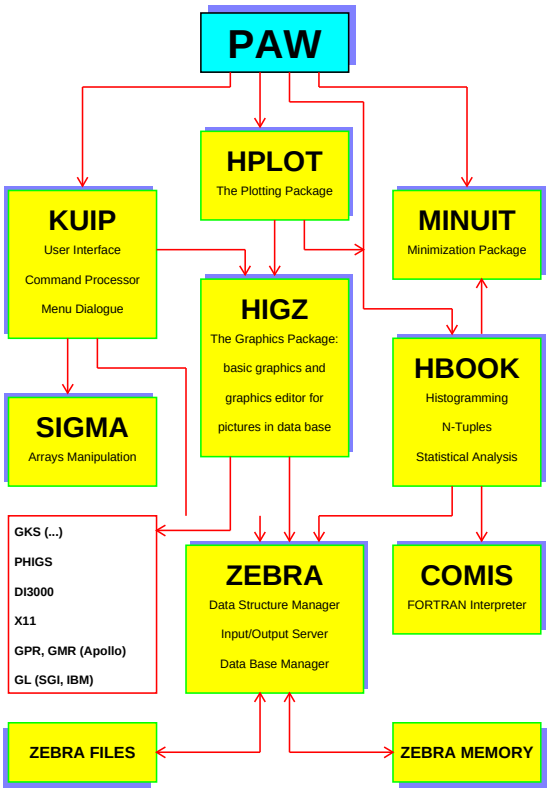


Figure 8.3: PAW and its components

- Fit a function to a histogram
- Annotate and print graphics

8.3.2 `cmz`

`cmz` is an advanced, interactive, fast, self-documenting, customizable, machine-independent and `patchy` compatible source-code management system with emphasis on FORTRAN and C source code. `cmz` is based on the same user interface package as PAW (KUIP). It allows to develop and transfer machine independent code for application programs.

The source-code management for almost all packages of the CERN program library is done with `cmz`, so it is a useful tool to develop user applications for different hardware platforms in the CERN environment. `cmz` runs on all supported UNIX flavors as well as on IBM/MVS, VM, VMS and other operating systems.

Users intending to use `cmz` for their code development should consult the printed manual.

8.3.3 GEANT

GEANT is a system of detector description and simulation tools which should help the physicist in such studies. GEANT is most useful in the

- design and optimization of the detector
- development and test of the reconstruction and analysis programs, and
- the interpretation of experimental data.

In order to run GEANT, the user has to provide his own set of standardized subroutines describing the geometrical setup, the kinematics of the event and storing the results into the desired histograms.

GEANT can be run in interactive mode with graphics capabilities for program development and detector setup and, if settled, in batch mode for production of statistical data.

8.3.4 Organization of the CERN program library

The CERN program library is available in machine independent source (`car` file) and machine dependent source and binary form. Usually, CERN offers updates of the library 3-4 times a year. Three different versions of the CERN-library are stored on every machine:

- `pro`: production version
- `new`: newest version, may be changed without notice.
- `old`: old version, you can fall back to this version in case of incompatibilities.

The CERN library is stored in a standardized place: You will find all files in the directory `/cern`. The organization of the subdirectories is shown in table 8.1

The libraries listed in table 8.2 have been installed in the `lib` subdirectory.

8.3.5 Usage of CERN-library programs

Initialization of CERN environment

Before you can use any program of the CERN library, you have to initialize the CERN environment. This can be done by the command

```
. cernlogin [new | pro | old ]
```

If none of the optional parameters is given, the `pro` version is selected.

You can initialize the CERN environment automatically during login. To do so, put the following line (as single line) into your `$HOME/.profile` file (it should be already in the file, just remove the comment sign # in front of the line):

cernlogin in .profile

```
[ -x /usr/local/bin/cernlogin ] &&
 . /usr/local/bin/cernlogin
```

If you login on a HP machine via VUE-login, remember that the `.profile` file *is not* executed, but instead the `$HOME/.vueprofile` is executed. In this case, put the following line (as single line) into your `$HOME/.vueprofile` file (it should be already in the file, just remove the comment sign # in front of the line):

cernlogin in .vueprofile

```
[ -x /usr/local/bin/cernlogin ] &&
 . /usr/local/bin/cernlogin >/dev/null 2>&1
```

Starting `paw`:

You can start `paw` after having initialized the CERN environment (see above) by typing

```
paw
 ******************************************
 *                                        *
 *       W E L C O M E    to   P A W       *
 *                                        *
 *      Version 1.13/00  9 March 1992     *
 *                                        *
 ******************************************
 Workstation type (?=HELP) <CR>=1 :?
 List of valid workstation types:
       0:  Alphanumeric terminal
    1-10:  Describe in file higz_windows.dat
  n.host:  Open the display on host(1<n<10)
       m:  PAW_MOTIF on local host
  m.host:  PAW_MOTIF on specified host
    7878:  FALCO terminal
    7879:  xterm

 Metafile workstation types:
    -111:  HIGZ/PostScript (Portrait)
    -112:  HIGZ/PostScript (Landscape)
    -113:  HIGZ/Encapsulated PostScript
  -777/8:  HIGZ/LaTex
```

`/cern`			main directory CERN program library
	`/old`		old version, and subdirs
	`/pro`		production version with the following subdirs:
		`/bin`	executables (`paw, cmz,`...)
		`/lib`	libraries (`libpacklib.a`,...,`gxint315.o`)
		`/src` `/car`	machine independent source (`car`-files)
		`/cra`	cradles, instructions how to unpack `car`-files
		`/mkf`	
		`/cfs`	`/package`: source for each package
		`/inc`	GEANT 3.15 include files
		`/log`	log-files of library installation
		`/mgr`	installation tools for library manager
		`/doc`	documentation for some pool-packages (not supported)
		`/pawexam`	paw-example `kumac` files from paw-manual
	`/new`		new version, and subdirs
	`/doc`		postcript versions of manuals and CNL

Table 8.1: structure of the CERN program library

name	**library**	
KERNLIB	`libkernlib.a`	basic mathematical and general purpose routines
PACKLIB	`libpacklib.a`	main packages (CSPACK, EPIO, FFREAD, HBOOK4, KAPACK, KUIP, MINUIT, ZBOOK, ZEBRA) and duplication of KERNLIB
GRAFLIB	`libgraflib.a`	grafic packages (HPLOT5, HIGZ), kernel
	`libgrafX11.a`	X11-part of GRAFLIB
	`libgrafGKS.a`	GKS-part of GRAFLIB
PAWLIB	`libpawlib.a`	PAW-libraries (PAW, COMIS, SIGMA)
GEANT	`libgeant314.a`	GEANT library version 3.14 (only for backwards compatibility)
	`libgeant315.a`	GEANT library version 3.15
	`geant314.o`	interactive main program GEANT 3.14
	`geant315.o`	interactive main program GEANT 3.15
BVSL	`libbvsl.a`	bitvector manipulation package
GENLIB	`libgenlib.a`	general purpose subroutines

Table 8.2: CERN libraries installed on `/cern/`*version*`/lib` subdirectories

After the start-up, `paw` asks for the workstation type. A question mark (?) gives you a list of valid workstations. Usually you enter either `0` for no graphics or a number between 1 and 10. You can customize the initial appearance of your graphics window by editing the file `$HOME/higz_windows.dat`. Each of the 10 lines in this file corresponds to one workstation number and describes the position and size of the graphics window.

The following additional parameters can be given to the `paw`-command:

option	
`-v [new|old|pro]`	version, default is selected version of `cernlogin`
`-n`	disable automatic execution of `pawlogon.kumac`
`-b` *macroname*	batch mode, execute *macroname*, implies workstation type = 0

Starting `cmz`:

`cmz` is started after initialization of the CERN environment (see above). The following additional parameters can be given to the `cmz`-command:

option	
`-v [new|old|pro]`	version, default is selected version of `cernlogin`
`-n`	disable automatic execution of `cmzlogon.kumac`
`-l` *logon*	read logon commands from file `logon.`*kumac*
`-r [`*restore*`]`	restore environment of previous CMZ session. By default, `cmzsave.dat` will be read, if specified file *restore*
`-b` *macroname*	batch mode, execute *macroname*, implies also `-n`

Creating your own applications

To compile and link your own Fortran or C programs together with routines from the CERN-library, you can use standard Unix compile or link statements (see chapter 7 on page 45). The use of a makefile is strongly encouraged (see section 7.4.1 on page 46).

ATTENTION: You *must* specify the following compile options, depending on the operating system of your machine, otherwise no CERN library subroutine can be found by the linker:

option	OS
`-qextname`	AIX
`+ppu`	HP-UX
	DEC-ULTRIX

The following examples show two Makefiles for creating

1. a GEANT program
2. a PAW application with own extensions

A GEANT example:

The following `Makefile` shows an example of how to create the interactive GEANT application `shower` from the user supplied subroutines `gudigi.f, gufldi.f, gukine.f, ....` These routines use `INCLUDE` statements to include the standard GEANT common blocks from the directory `/cern/pro/src/inc`.

The following `Makefile` can be used on an HP machine. You can copy this `Makefile` to create your own from `/cern/doc/Makefile.geant`.

GEANT Makefile HP-UX example:

```
# GEANT Makefile example
# M.D. 15/10/92
#
# definition of symbols
#
LIB=/cern/pro/lib
INC=/cern/pro/src/inc
#
# definition of own modules
#
GEANTOBJ=gudigi.o gufldi.o guffgo.o gufld.o \
         gugeom.o guhbook.o gukine.o \
         gumate.o gustep.o
#
# Default action
#
.DEFAULT: shower
#
# Compiler options
#
# for debug:
# FOPTS=-g +ppu
# for production:
FOPTS=+O +ppu
```

```
FFLAGS=$(FCOPTS) $(FOPTS)
#
# general rule how to compile .f
#
.f.o:
fort77 $(FFLAGS) -c -I$(INC) $<
#
# rule how to create program shower
#
shower: $(LIB)/gxint315.o $(GEANTOBJ)
fort77 $(LIB)/gxint315.o $(GEANTOBJ) \
-L$(LIB) -L/usr/lib/X11R4 \
-lgeant315 -lgraflib -lgrafX11 -lpawlib \
-lpacklib -X11 -lm \
$(FFLAGS) -o shower;
```

If you are running on an IBM RS/6000, the following lines have to be changed:

GEANT Makefile AIX example:

```
⋮

# FCOPTS=-g -q extname
# for production:
FOPTS=+O -q extname
⋮

.f.o:
xlf $(FFLAGS) -c -I$(INC) $<
#
# rule how to create program shower
#
shower: $(LIB)/gxint315.o $(GEANTOBJ)
xlf $(LIB)/gxint315.o $(GEANTOBJ)
⋮
```

For an explanation of the `-I, -c, -l, -L, -o` flags on the compile command, see the man pages and section 7.2 on page 45 of this primer.

A PAW example:

The following `Makefile` shows an example of how to create the interactive PAW application `pawgr1` with user supplied commands. The new commands are defined in the command definition file (`cdf` file) `user.cdf`. They call action routines from `condset.f, condshow.f, ...`.

Additionally, the user written subroutines are archived in a library `libpawgr.a`. Object files are archived in this library by the command `ar`.

The last entry of the `Makefile` shows how to create automatically the appropriate fortran routine for command definition – which has to be called in the PAW main program – from the `cdf`-file by calling the KUIP-compiler `kuipc`.

The following `Makefile` can be used on an HP machine. You can copy this `Makefile` to create your own from `/cern/doc/Makefile.paw`.

PAW Makefile HP-UX example:

```
# PAW Makefile example
# M.D. 15/10/92
#
# definition of symbols
#
LIB=/cern/pro/lib
#
# definition of own modules
#
PAWOBJ= condset.o \
        condshow.o\
        conddraw.o\
        user.o
PAWMAIN=mypaw.f
#
# Default action
#
.DEFAULT: pawgr
#
# Compiler Options etc.
## for debug:
# FOPTS=-g +ppu
# for production:
FOPTS=+O +ppu
FFLAGS=$(FCOPTS) $(FOPTS)
ARFLAGS=-crv
#
# general rule how to compile .f
#
.f.o:
fort77 $(FFLAGS) -c $<
#
# rules how to create program pawgr1
#
pawgr1: $(PAWMAIN) libpawgr.a
fort77 $(PAWMAIN) \
-L. -L$(LIB) -L/usr/lib/X11R4 \
-lpawgr -lpawlib -lgraflib -lgrafX11 \
-lpacklib -X11 -lm \
$(FFLAGS) -o pawgr1

libpawgr.a: $(PAWOBJ)
echo "Loading libpawgr.a ..."
ar $(ARFLAGS) libpawgr.a $(PAWOBJ)
#
# create user.f from command
# definition file user.cdf
#
user.f: user.cdf
kuipc user.cdf user.f
```

If you are running on an IBM RS/6000, the same changes as above in the GEANT example have to be done.

For an explanation of the `-I, -c, -l, -L, -o` flags on the compile command, see the man pages and section 7.2 on page 45 of this primer.

Chapter 9: Introduction to Internet Services

9.1 About Internet

The Internet is a global network of networks that provides access to hundreds of thousands of computers around the world. As the reach of the network has grown, so has the number of services accessible. the main tools that allow the user to navigate through the Internet, are

`telnet` to access remote hosts,
`ftp` to retrieve data files.

Anonymous FTP will be briefly described next. For `telnet` see chapter 4.4 on page 23.

9.1.1 FTP

FTP stands for "file transfer protocol" and is the method used to transfer files over the Internet. "Anonymous" ftp means that one can login to the remote system using the userid of "anonymous" and password of either "guest" or usually your own userid and internet address. Ftp is like telnet in that the "open" command and access to the remote host is similar.

A typical session might go as follows:

```
$ftp any.host.i.know
login:anonymous
guest login ok...send user id as password
ftp>ls -al  (list all files)
ftp>cd pub (change to the " pub" directory)
ftp>get my.file
transfer complete
ftp>quit
$
```

The standard transfer protocol is ASCII. This is suitable for text. Use command `binary` if transferring program or image files. (Note: on VAX-VMS computers use `IMAGE`).

Large files are usually "tared" and compressed. You have to use binary FTP to get such files. The file extension shows how to uncompress it:

`.tar` `tar-xvf` *myfile.tar*
`.Z` `uncompress` *myfile.Z*
`.tar.Z` `uncompress` *myfile.tar.Z*
`tar -xvf` *myfile.tar*

9.1.2 Internet addresses

There are two forms that express an Internet address, an alphabetic name, or a series or numbers. The alphabetic version is called the "domain name system" and the numeric the "numeric name system". Sometimes a local network will not be up-to-date with additions to the domain names and an address may not work. If this happens, try the numeric address before giving up. Sometimes the numeric name system address will be changed without notice and in that case the alphabetic domain name should be tried.

9.2 Internet Services

9.2.1 Overview

With a little practice, the above-described functions (ftp,telnet) will be simple and open the electronic door to the global reach of the Internet. An introduction to the Internet services can be found in [10] A comprehensive listing of services is given in [14]. Its table of contents is listed below:

1. Library Catalogs & Campus Information Systems
2. Databases
3. Electronic Discussion Groups/Forums
4. Directories
5. Information Resources
6. FTP Archives
7. Fee-Based Information Services
8. Software/Freeware
9. Bulletin Board Services
10. Miscellaneous

9.2.2 archie

One of the most useful Internet services, acquisition of public domain software, can be the most frustrating. There are now hundreds of servers with thousands of software titles spread throughout the Internet. Often the searcher knows that the needed software is somewhere out there but finding the software title through this maze can take a long time. After checking 10 or 20 host sites, one is tempted to give up. Archie is a unique system devised to make locating software on public archives simple. Instead of searching the remote hosts one at a time, the user can enter the search on "archie" and find out where copies exist across 712 (at this writing) hosts.

The results of the search may be viewed online or sent automatically via e-mail for later viewing. Search results identify host domain name and IP address and the exact path and filename to the requested file making it easy to ftp. The search engine has many powerful features to aid in retrieving those hard- to-find titles.
Access: `telnet quiche.cs.mcgill.ca` login as `archie`. Important commands are:

`help`	a list of all commands
`help command`	description of command
`quit`	exit archie
`whatis`	search for keyword in the software description database
`prog`	search the database for a file

For example `prog xclock` will cause archie to search all the archives for the string "xclock". At the end of the search, archie will present the results back to the screen.

`xarchie`

If you have access to an X-window terminal, you can use `xarchie` for a menu-guided search through the public-domain ftp sites. For a more detailed description use `man xarchie`.

9.2.3 NetNews

Usenet is the set of people who exchange articles tagged with one or more universally-recognized labels, called "newsgroups". The groups distributed worldwide are divided into seven broad classifications: "news", "soc", "talk", "misc", "sci", "comp" and "rec". Each of these classifications is organized into groups and subgroups according to topic.

comp Topics of interest to both computer professionals and hobbyists, including topics in computer science, software source, and information on hardware and software systems.

sci Discussions marked by special and usually practical knowledge, relating to research in or application of the established sciences.

misc Groups addressing themes not easily classified under any of the other headings or which incorporate themes from multiple categories.

soc Groups primarily addressing social issues and socializing.

talk Groups largely debate-oriented and tending to feature long discussions without resolution and without appreciable amounts of generally useful information.

news Groups concerned with the news network and software themselves.

rec Groups oriented towards hobbies and recreational activities.

To start a NetNews reader with X11 interface type `mxrn` on the IBM or HP workstations.

Appendix A: GNU Software

GNU stands for **Gnu's Not Unix** and is the name for the complete Unix-compatible software system developed by the Free Software Foundation. Some large parts of this system are already working, and are distributed now.

The Software is distributed as a 'Free Software'. The word 'free' stands here for freedom and not to price. To get the GNU Software you may pay or may not. But in contrast to commercial Software, you have the freedom to copy the program and give it to your friends and co-workers and you have the freedom to change the program as you wish, by having the full access to the source code. Furthermore, you can study the source and learn how such programs are written.

The main work of the Free Software Foundation is concentrated on the development of new free software, working towards a complete GNU system.

Beside developing GNU, FSF distributes copies of GNU software and manuals, and accepts tax-deductible gifts to support GNU development. Most of the FSF's funds come from its distribution service.

Several parts of the GNU Software (about 120 programs) are available for UNIX, DOS, and a lot of other operating systems.

Interesting GNU programs for a UNIX system are:

emacs	full-screen editor (Chapter 5.2)
gcc	the GNU C/C++/Objektiv-C Compiler
gdb	GNU C,C++ Debugger
GNU C Lib	POSIX.1 C library
libg++	C++ class library
bison	GNU advanced yacc
make	GNU advanced make
indent	C reformatting program
RCS	Revision Control System
CVS	Concurrent Version System
patch	apply diffs (patches) to files
gnuchess	a chess playing program (X11)
Ghostscript (gs)	a Postscript interpreter
ghostview	X11 user interface for Ghostscript
fontutil	fonts for Ghostscript or TEX
gnuplot	interactive plotting program
Texinfo	structured documentation system, produces on-line help and printed documents
ispell	advanced spell checker (also for TEX)
less	a better pager
bash	GNU's Bourne Again SHell
GNU tar	GNU (Tape) archive
patch	applies diff files

We have installed some of the GNU programs in the `/usr/local` filesystem on the central AIX and HP-UX file servers, so that the programs are available via NFS.

Nearly all of the above listed programs are installed on the central AIX server.

For the most programs exist a man page or a Postscript file of the reference manual in `/usr/local/doc/gnu`.

All the software and publications from the Free Software Foundation are distributed with permission to copy and redistribute. If you are interested in a copy you can get the latest software via anonymous FTP (program: 'ftp', user: 'anonymous', password YOUR NAME, mode 'binary') from prep.ai.mit.edu (18.71.0.38) (Directory: `/pub/gnu`). This ftp server is in Cambridge. To reduce the transfer time and costs you should take a ftp server at your side. Two good candidates are ftp.th-darmstadt.de (130.83.55.75) or rusmv1.rus.uni-stuttgart.de (129.69.1.12).

If you find a GNU program of common interest, and you think it should be installed on a central file server, let use know about it.

From our own experience: Please restrict your self in collecting programs from ftp servers. The disk space is restricted and it makes no sense that every body has the same programs on his user disk.

Appendix B: Motif Windows

With a mouse, a lot of functions can be performed with windows:

1. You can accomplish **geometrical** operations, such as moving or resizing.
2. The contents of windows can be **scrolled**.
3. Windows can be **iconized**, that means, they can be converted to a small picture called **icon**, which is a representation of an inactive window, and they can be restored again into the state of an active window.

Table B.1 summarizes the operations possible with Motif windows. The terms used there are explained in the following.
The remainder of this paragraph describes in detail, how these operations can be performed.

Standard GSI Terminal Windows

A standard GSI terminal window can be obtained with the command `xt`. It is an Motif window and has an outer **frame** and a horizontal **title bar** on top and a vertical **scroll bar** at the right, but both within this frame. They consist of several parts with each of them enabling specific functions. The mouse cursor, which has the shape of an 'I' when positioned within the text field, changes its appearance when moved to one of these elements. The new shape depends on the specific location.

Motif Window Frame

The Motif window frame - as a rectangle - consists of two horizontal and two vertical bars, and of four corners.
When moved to one of the bars, the mouse cursor appears as an arrow pointing outwards to a line. This shall indicate that the corresponding border of the window can be moved. To do it, press the left mouse button and move the mouse cursor while keeping the button pressed: The window is resized by **dragging** this border, whereas the other borders keep their location.
When moved to one of the corners, the mouse cursor appears as an arrow pointing outwards to the corresponding window corner. In the same way as with the bars, the window can be resized by dragging this corner, whereas the diagonal opposite corner keeps its location.

Title Bar

The title bar is the horizontal bar on top of the screen, just inside the window frame. It consists (from left to right) of the window menu button, the title area, the minimize button, and the maximize button.

The Title Area If you move the mouse cursor to the long horizontal field of the title bar, the cursor changes its shape to a fat arrow. Now, with the same mouse procedure as described above, you can move the window as a whole without changing its size.

The Minimize Button The minimize button can be used to convert a window to an icon. The button is identified by a very small square. To **iconify** a window might be useful in case of programs that can run unattended in background, or in case of applications used only occasionally. It preserves screen space by reducing the window stack.

The Maximize Button The maximize button can be used to convert an icon or a window to a new window covering the whole screen. It is identified by a big square in its center.

The Window Menu The window menu button is located leftmost within the title bar and identified by a narrow rectangle in its center. If activated with the left mouse button, either by clicking or pressing and holding down, the window menu appears as popup menu. It can be used for the handling of windows and also of icons - the window menu also pops up if you click an icon. The functions not available are printed in a lighter typeface (minimize and size in the case of an icon). All functions available with the mwm window frame and the other parts of the title bar are also available with the window menu.

If a window menu function is activated by click with the left mouse button, it can be performed just by shifting the mouse - no more button needs to be pressed during mouse movement. The following functions are available in the window menu:

- **Restore**: Icons, or windows covering the full screen ('maximized'), are converted back to a window.
- **Move**: Move window or icon with the mouse.
- **Size**: Resize window. If you move the mouse to one of the four corners of the window frame, the corresponding corner points are shifted. If you move the mouse to one of the four vertical or horizontal borders, the corresponding borders are shifted.
- **Minimize**: Convert window to icon.

Operation	window part	action
resize window	window frame	drag vertical or horizontal **bar** (top or bottom)
	window frame	drag **corner** (one of four)
	window menu	click **size** button, shift frame bar
	window menu	click **size** button, shift frame corner
move window	title bar	drag **title area**
	window menu	click **move** button, shift window
iconize window	title bar	click **minimize** button
	window menu	click **minimize** button
maximize window	title bar	click **maximize** button
	window menu	click **maximize** button
restore window from icon	window menu (icon)	click **restore** button
lower window in stack	window menu	click **lower** button
close window	window menu	click **close** button
move icon	window menu (icon)	click **move** button, shift icon
maximize icon	window menu (icon)	click **maximize** button
lower icon in stack	window menu (icon)	click **lower** button
close icon	window menu (icon)	click **close** button

Table B.1: Summary of Operations with Motif Windows and Icons

- **Maximize**: Convert window or icon to window covering the complete physical screen.
- **Lower**: Put window or icon to the bottom of the window or icon stack.
- **Close**: The window will be closed (and disappears).

The window menu functions can also be invoked via accelerator keys, e.g. **`<Alt-F9>`** for Minimize. This works with or without an activated window menu. If the window menu is already activated, you can also enter a unique abbreviation (the underlined letter), e.g. 'n' for Minimize. The required key combinations and abbreviations are indicated in the window menu.

The Scroll Bar

In your terminal windows your are not limited to the displayed lines of text. When created with `xt`, up to 200 lines of text are saved, and you can use the scroll bar at the right to scroll through the saved text.

Appendix C: Unix Hardware at GSI Computing Center

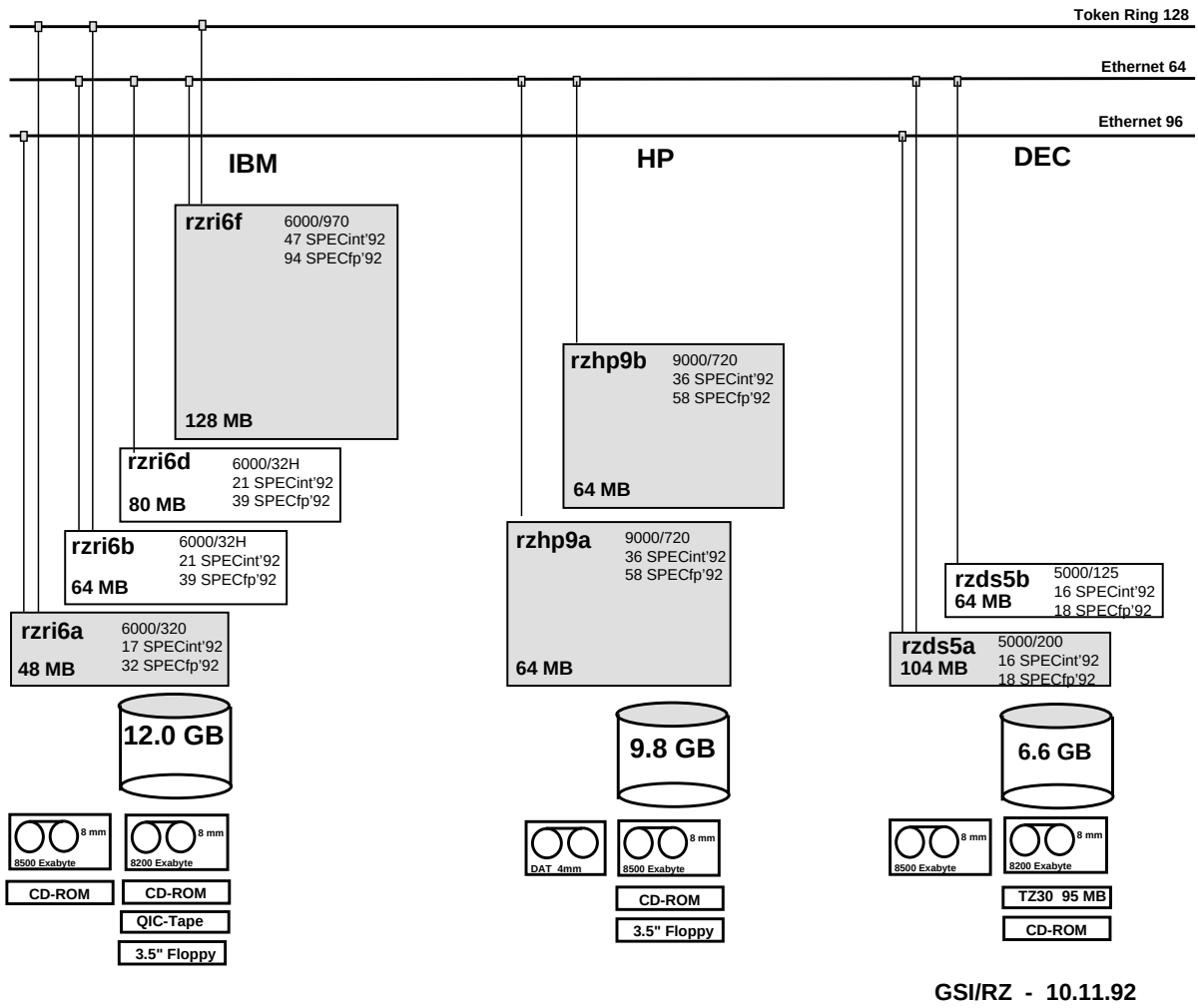


Figure C.1: Present status of the Unix Hardware installed at GSI Computing Center. Grayshaded boxes represent workstations accessible by users.

Bibliography

[1] Adobe Systems Incorporated, *The PostScript Language Reference Manual*, Addison-Wesley, (1990), ISBN 0-201-18127-4

[2] D. Cameron and B. Rosenblatt, *Learning GNU Emacs*, O' Reilly &Associates, Inc, Sebastopol, USA,(1991)

[3] J.D. Foley, A. van Dam, S.K. Feiner, J.F. Hughes, *Computer Graphics: Principles and Practice*, Addison-Wesley, Amsterdam, (1990), ISBN 0-201-12110-7

[4] M. Goossens, A. Samarin, *TEX at CERN - Local Guide*, CERN CN/US/136, (1992)

[5] Hewlett-Packard Company, *The Ultimate Guide to the vi and ex Text Editors*, Benjamin/Cummings Publishing Company, Inc., Redwood City, USA, (1990)

[6] T.L.J. Howard, W.T. Hewitt, R.J. Hubbold, K.M. Wyrwas, *A Practical Introduction to PHIGS and PHIGS PLUS*, Addison-Wesley, Amsterdam, (1991), ISBN 0-201-41641-7

[7] R.O. Jenks, R.S. Sutor, *axiom, The Scientific Computation System*, Springer

[8] D.E. Knuth, *The TEXbook*, Addison-Wesley, Reading, (1990)

[9] D.E. Knuth, *Computers and Typesetting*, Vol. A – E, Addison-Wesley, Reading, (1986)

[10] E. Krol, *The Whole INTERNET, Users's Guide and Catalog*, O' Reilly, (1992), ISBN 1-56592-025-2

[11] L. Lamport, LATEX, *A Document Preparation System*, Addison-Wesley, Reading, (1986)

[12] L. Lamb, *Learning the Vi*, O' Reilly &Associates, Inc, Sebastopol, USA, (1990)

[13] F. Mittelbach, R. Schöpf, *The New Font Selection - User Interface to Standard* LATEX, TUGboat 10,2 (1989) 222-238

[14] *NYSERVNet, New User's Guide to Useful and Unique Resources on the Internet*, stored in `rzri6b:/usr/local/doc/internet/nysernet.guidev2.txt`

[15] St. Talbott, *Managing Projects with make*, O' Reilly &Associates, Inc

[16] St. Wolfram, *Mathematica, A System for Doing Mathematics by Computer*, Addison Wesley

Index