Unix Basics

Revision: March 18, 2005

Author: Mark Könnecke

Contents

1 Introduction **3**

2 Unix at NUM **3**
2.1 NUM Linux Computers . . . 3
2.2 Important Linux Servers running SL3.0 . . . 4
2.3 PSI Unix Accounts . . . 4

3 Logging In and Out **5**
3.1 Logging in at a Unix Workstation . . . 5
3.2 Logging in to a remote Unix Workstation . . . 5
3.3 AFS Logins expire after 25 Hours! . . . 6

4 The Linux KDE Desktop **6**

5 Unix Shells **6**
5.1 Command Line Editing . . . 6
5.2 Getting Help . . . 7
5.3 Running Programs . . . 7
5.4 File Operations . . . 9
5.4.1 File Specifications . . . 9
5.4.2 File Manipulation . . . 9
5.4.3 Directory Manipulation . . . 10
5.5 Printing from Unix . . . 10
5.6 Environment Variables . . . 11
5.7 Input- Output- Redirection . . . 11
5.8 Common Commands . . . 12

6 Editing under Unix **13**
6.1 Editing with jed (and emacs) . . . 13
6.2 Editing with edt or xedt . . . 14

7 Command Files or Shell Scripts **15**
7.1 The login script . . . 16

8 The AFS Filesystem **16**

9 Frequently Occuring Problems **17**

10 Programming under Unix **19**
10.1 Compiling . . . 19
10.2 Compiling and Linking in one Go . . . 19
10.3 Linking . . . 19

10.4 Linking with Libraries 19
10.5 Building programs with Make 20
10.6 Debugging Programs 21
10.6.1 Using core Files 22

11 Unix in the SINQ Hall **22**
11.1 Local Accounts 22
11.1.1 Local Accounts on Instrument Computers 22
11.1.2 Linux Clients in the SINQ Hall 22

12 Windows and Unix **23**
12.1 Accessing Unix from Windows 23
12.1.1 Text Mode Login to Unix from Windows 23
12.1.2 Graphical Login to use X–Windows from Windows–PC 23
12.2 Data Exchange between Windows and Unix 23

13 Macintosh and Unix **24**
13.1 Accessing Linux Computers from Macintosh 24
13.2 Data Exchange between MacOS X and Unix 24
13.3 File Exchange from MacOS X to Windows 24

1 Introduction

Unix is a very powerful and fast operating system and is the only serious competitor to the Microsoft world left in the operating system market. It is also very old: Unix dates back into the 1970s. What could be considered a weakness is actually a strength: unix has been flexible enough to accomodate 30 years worth of hardware and software development and is very likely to continue to do so. The faint hearted are put of by unix because of its sometimes cryptic command language and its complexity. However with an inquisitive mind and playfullness the power of unix can be easily mastered. Eventually you may realise that cryptic commands actually save time because you do not need to type so much. Unix IS user friendly, it may just be a bit particular about who its friends are! The complexity arises from the fact that a modern operating system provides so many services to its users. In contrast to Windows, the configuration of such services is always accessible under unix. The purpose of this document is twofold:

- It gives an overview of the use of unix at NUM
- It introduces new users to the most important unix concepts and commands.

2 Unix at NUM

Unix at NUM is Linux. The Linux distribution used is Scientific Linux 3.0.x (SL3.0). Scientific Linux is Redhat Enterprise server Linux recompiled from sources by CERN and FERMILAB. These laboratories also added some scientific software to the default distribution. There are also a couple of alpha stations running Tru64Unix 5.1 but these are falling into disuse and are maintained for reasons of backward compatibility only.

2.1 NUM Linux Computers

The standard PSI distribution of Scientific Linux comes in two different forms:

Green PC's This kind of distribution is automatically maintained and updated by PSI computing staff. This implies that even NUM computing staff do not have root access to such computers. There is a local administrator account on such machines however which allows to perform certain operations. in NUM Green PS's are all user Linux PC's and the client computers in the SINQ hall.

Red PC's For red PC's it is the users responsability to perform system maintainance and upgrades. To this pupose the user has root access. In NUM all instrument computers and the servers are red PC's.

Not all users with a PSI AFS account may login to NUM Linux systems, even when using ssh. For personal linux machines, only the main user may login. Access to servers and instrument computers is allowed to LNS group members and assorted local user accounts only. Please keep this in mind when collaborating with external users or PSI staff from alien departments.

2.2 Important Linux Servers running SL3.0

This section contains a list of important Linux PC's.

- hrpt.psi.ch, the HRPT instrument computer
- dmc.psi.ch, the DMC instrument computer
- sans.psi.ch, the SANS instrument computer
- sans2.psi.ch, the SANS2 instrument computer
- tasp.psi.ch, the TASP instrument computer
- poldi.psi.ch, the POLDI instrument computer
- morpheus.psi.ch, the MORPHEUS instrument computer
- rita2.psi.ch, the RITA-2 instrument computer
- trics.psi.ch The TRICS instrument computer
- focus.psi.ch The FOCUS instrument computer
- amor.psi.ch The AMOR instrument computer

There are some central Linux servers which are to be used for CPU or data intensive data analysis tasks:

- lnsl15.psi.ch, the NUM Linux Science Server
- llc.psi.ch, the PSI Linux cluster
- lns00.psi.ch The NUM WWW–server

2.3 PSI Unix Accounts

Each user on a unix computer has a home directory assigned to her. This is a space on the harddisk where the users data is stored. You may now start to wonder, how you should distribute your data among all these computers. The answer is that this is not necessary. NUM users have **network user accounts**. This means that your user credentials are valid on all machines. Your data is physically stored on a central server and provided to you wherever you login through a magical mechanism called **network file system**. The network file system in use at PSI (and at NUM) is the **Andrew File System (AFS)**.

AFS has some nice features:

- Data on AFS is visible to Windows or Macintosh PC's as well, if the appropriate AFS client software is installed. Do not worry, this software has already been installed for you on any PSI PC.
- Data on AFS filesystems is available to you worlwide on any computer which has AFS–client software installed.

- AFS uses caching. This means a file from the remote server is downloaded to a local cache area on first access. After that any further access is as fast as a local access.
- The PSI computing center takes care to backup your data on AFS.

There are some important paths strings to remember when working with AFS. The first one is your home directory: Home directoy names are built according to the following scheme:

`/afs/psi.ch/user/first_letter_of_the_username/username`

An example:

`/afs/psi.ch/user/h/heer`

This is the home directory of the user Heer.

Then there are common directories:

/afs/psi.ch/project/sinq and sub hierarchy contains programs and libraries provided by NUM staff for common usage.

/afs/psi.ch/project/sinqdata is the path to the SINQ data files. Below this path are further subdirectories for each year of SINQ operation and in the year directories, directories for each instrument.

Normal users can read and execute everything on these common areas. But only NUM computing staff has write privilege. But for larger software projects NUM computing staff can enable write access to designated directories.

3 Logging In and Out

There are several different ways for getting into one of the NUM unix machines. Two different cases have to be distinguished:

- Logging in directly to a unix workstation.
- Logging in to a remote unix workstation.

For information how to connect to unix computers from Windows– or Macintosh–PC"s consult the appropriate sections at the end of this paper. At NUM the standard login shell is the **tcsh**.

3.1 Logging in at a Unix Workstation

The first one is the easiest: It applies when you walk up to a unix workstation. You will see a window with a central panel prompting you for your username and password in succession. Then the KDE desktop will start up.

3.2 Logging in to a remote Unix Workstation

The recommended way to access other unix computers when already logged in to a unix system is to use the command **ssh -X user@hostname**. You will be prompted for a password. The –X causes any graphical X11–application to display on your monitor.

3.3 AFS Logins expire after 25 Hours!

AFS logins have a security feauture: After 25 hours they expire. This means if you stayed logged in over night your account will cease to function. Symptoms are: you cannot write data to your home directory, you cannot start programs. Logging out and in again will solve the problem. There are a few useful commands to help with this:

tokens shows you the expiry date of your login.

klog username asks for your password and then goes away and refreshes your login (or in AFS–speak, your token).

4 The Linux KDE Desktop

On Linux there is the KDE desktop. On the bottom of the screen there will be a toolbar containing several icons. The most important of this is the cog wheel with the K in it at the left. This icons opens a Windows–95 look alike menu of programs which can be started. In this menu you also find options to logout or lock the desktop.

Another important icon is the computer screen symbol with the shell. Clicking this opens a terminal window.

Then there is an icon looking like a lifebuoy. This opens the KDE help system which may be consulted for more details on KDE.

In the middle there are fields labelled *one, two, three, four*. Now, KDE supports the virtual screen concept. A virtual screen can be understood as a monitor realized in software. The purpose is to have multiple work areas on the same physical screen. This comes in handy when working at different projects at the same time. The buttons permit to switch between these virtual screens.

5 Unix Shells

Once logged in to a unix system in text mode or after opening a terminal window you will encounter the shell. This is the command line interface for user interaction with unix. Unlike VMS, which supports only one shell (DCL), the shell is just a normal program to unix. This lead to the development of not just one but several shells for unix, a fact which might confuse faint-hearted users. The most common shells are: sh, bash, ksh, csh and tcsh. NUM has standardized on the tcsh for interactive work as it has the best command line editing features of the lot.

A special note to former VMS or MS-DOG users: **All input to unix is case sensitive!**.

5.1 Command Line Editing

The content of the current comand line can be manipulated with the special keys or key combinations listesd below:

Arrow Up Scrolls up one line in the command history.

Arrow down Scrolls down one line in the command history.

TAB completes a command or filename if enough characters have been typed to make the name unique.

BACKSPACE deletes the last character typed.

Right- (Left) Arrow move right or left in the current line.

Ctrl-a move to beginning of line.

Ctrl-e move to end of line.

\ at the end of the line allows to continue the command on the next line. This is useful for those extra long commands and filenames.

5.2 Getting Help

man-pages For each unix command there is a man page describing it. A **man** page is unix speak for a manual page. Two commands are important:

man progname prints help for the command progname.

man -k keyword searches all available man pages which contain the keyword specified. This is useful if a command name is unknown.

The displayed page can be scrolled by hitting the **space** key. Viewing can be stopped by hitting the key **q**.

WWW Relevant man pages for Linux can be accessed through links on the NUM computing homepage, http://lns00.psi.ch

Paper A set of printed documentation is available in WHGA/112. A general introduction to unix is available as a book from Heinz Heer.

This document Updated versions of this document may be available at the URL:

`http://lns00.psi.ch/misc/unixbasic.pdf`

5.3 Running Programs

program A program is started in unix by just typing the program name.

Ctrl-C stops an interactively running program in most cases.

Ctrl-Z suspends an interactively running program. Then you have two options:

bg makes the suspended program run in the background as if it had been started with **progname &**.

fg continues the execution of the suspended program in the foreground. This means a program can be suspended with Ctl-Z, some other operation can be performed and then the suspended program can be called back with *fg*.

Suspended programs can only be killed by bringing them into the foreground again and stopping them properly. Or use the **kill -9** scheme described below.

program & Appending a space and an ampersand to a program name runs the program in the background. There is no elaborate system of batch queues under unix. Depending on the shell background programs continue to run after logout. If this fails, try to start the program with **/usr/bin/nohup** *program* **&**. This makes absolutely sure that the program continues even after logout. Make sure that programs running in the background like this do not need input or output or have their input or output properly redirected (see below). Otherwise the program may not behave as expected. This only works if your program runs less then 25 hours. See aexec below for longer running programs.

aexec -t refresh-intervall -u afs-user -f pwdfile -bg command parameter If your program runs longer then the lifetime of your AFS token (25 hours), it will be killed. Aexec now refreshes your token at certain intervalls and your program keeps running. The parameters are:

-t refreshintervall The time between refreshes of the AFS–login in second.

-u afs-user The AFS username under which the program runs.

-p pwdfile A file which holds the AFS password of the user. This must be a file on a local file system (for example under /scratch) and may not be a file on a network file system

command parameter Are the actual program to run and its parameters.

For example:

```
aexec -t 7200 -u koennecke -p /scratch/mk.txt -bg anatric ok.xml
```

starts the program anatric with the parameter ok.xml. Anatric runs under the user koennecke as a background process. The AFS token will be refreshed any three hours. The password will be read from /scratch/mk.txt. Use aexec with greatest care! Storing your password in a file is a gaping security hole!

ps lists all processes started in the current session.

ps -A lists all process running on the computer.

ps waux lists all process running on the computer with full details.

kill -9 pid kills a runaway program. pid is the number appearing in the first field in a ps listing. Killing a program thus requires two steps:

- Locate the program ID with ps.
- Kill the program with: kill -9 pid

5.4 File Operations

5.4.1 File Specifications

Unlike VMS or DOS unix file systems do not support the concept of drives. All files are arranged in one huge directory tree. New disk partitions or network drives are added by mounting them at leaves of the root tree and thus making them part of the tree. The user is normally not aware of this. A note to VMS users: In a unix directory hierarchy one moves up into sub directories and down towards the root.

A fully qualified pathname to a file in a unix file system thus looks like this:

```
/afs/psi.ch/project/sinqdata/2000/focus/000/focus2000n000670.hdf
```

The directoy path is separated by / . At the end is the filename with an optional extension. There are NO VERSION NUMBERS in unix. However, many programs, notably editors, store the previous revision of a file as file~.

A full pathname can become quite long. There are various shortcuts: The first is the relative pathname:

```
../../bin/edj
```

This reads: move two directories down from the current position in the directory hierarchy, move into directory bin and fetch file edj.

Another shortcut is the specification relative to the users home directory:

```
~/bin/edj
```

This reads: start at the current users home directory, enter directory bin and fetch file edj. A reminder: the home directory is the directory you are in just after logging in.

Multiple files can be addressed through *wildcards*. The shell supports two wildcard characters:

***** is a substitute for 1-n arbitrary characters.

% is a substitute for a single arbitrary character.

An example: The command:

```
rm *.dat
```

deletes all files ending with .dat in the current directory.

5.4.2 File Manipulation

cp file1 file2 copies file1 to file2.

cp file1 dirname copies file1 into the directory dirname under the same name.

mv *file1 file2* renames (moves) file1 to file2

mv *file1 dirname* copies file1 into directory dirname and deletes the copy in the current directory.

rm *file1* deletes file1.

cat *file1* lists the content of file1.

more *file1* lists the contents of file1 in pages. Scroll the page by hitting the space bar, exit page viewing by hitting the q key.

grep blub *.dat searches all file matching the wildcard *.dat for the string blub and prints matching lines. Useful for searching based on file content.

5.4.3 Directory Manipulation

mkdir *dirname* creates the new directory dirname.

cd *dirname* changes the current working directory to dirname.

cd .. changes the current working directory to the one below the current one.

cd alone jumps to the users home directory.

pwd prints the full path of the current working directory.

rmdir *dirname* deletes the directory dirname. The directory must be empty.

rm -r *dirname* deletes the directory dirname with all contents. Use with care!

ls lists the contents of a directory.

ls -lisa lists the contents of the directory with all details.

find *startdir* **-name** *filename* searches the directory hierarchy for a file matching filename. The search starts at the directory startdir which must always be specified. If wildcards are used for filename the filename part must be enclosed in quotes.

mc starts a Norton Commander look alike text mode filemanager for unix.

5.5 Printing from Unix

All unix printing at PSI is now based on CUPS, the Common Unix Printing System. CUPS printer names start with the name of the building, followed by the number of the room closest to it, followed by an optional number. These three components are separated by underscores. Example: WHGA_U112_1 is a printer situated in building WHGA near room U112. A complete list of CUPS printers is available at:

http://cups.psi.ch/printers

A default printer can be assigned with the command:

lpoptions -d printername

Several commands exist for printing with CUPS:

xpp & starts a little graphical tools which allows to select a printer, a file to print and all options. Most comfortable.

kprinter is KDE's built in printer manager. Kprinter provides another graphical user interface to CUPS printers.

lp -d printer filename prints filename on printer from the command line.

lpstat -p -d lists all available printers on the command line.

lpstat printer shows job status for printer.

lpoptions -d printer sets default printer to printer.

Printing options can be specified to the **lp** command with *-o option*. For a list of options see:

http://cups.psi.ch/printpro-sum.html#STANDARD_OPTIONS

Commonly used options include:

-o sides=one-sided selects single sided printing

-o sides=two-sided-long-edge selects double sided printing.

-o media=A4 selects the paper to print on. Interesting choices are: A4, Transparency,MultiPurpose,U
or Lower.

-o cpi=12 sets the font size for printing.

-o lpi=6 sets the lines per inch.

5.6 Environment Variables

Unix supports the concept of environment variables. These are used to store some general system information or program information in a public space. The concept is similar to VMS logicals or symbols.

In the tcsh shell environment variables can be manipulated with the commands:

printenv *evar* prints the value of the environment variable evar.

printenv prints all environment variables.

setenv *evar newval* sets evar to newval.

printenv without parameters prints all environment variables.

5.7 Input- Output- Redirection

One of the strengths of unix is its powerful input/output redirection facility. This can be used as a simple scheme for interprocess communication and thus allows to concatenate various simple programs (filters in unix terminology) in order to achieve a complicated task. After initialization any unix program has three I/O channels to work with:

stdin The input read from the keyboard.

stdout Normal program output.

stderr Error output.

Each can be redirected. The syntax for this is:

program < *file1* lets program read its input not from stdin but from the file file1. This is useful for automating the running of interactive programs.

program > *file1* The output written to stdout from program is written into the file file1. After the program finished you may view the program output at your leisure by inspecting file1. This is very useful for programs like least squares programs which produce a lot of output. However, error messages printed to stderr still appear on the console.

program &> *file1* routes both standard program output and error messages into file1. This is useful for inspecting the errors generated by a compiler for a bad program.

program | **program2** | is the pipe symbol. The output of program is fed into the stdin channel of program2 for further processing. The usage is best described by an example: *ps waux* | *grep koenneck* for instance sends a full listing of all processes on the system into the program grep which selects only those lines containing the user name koenneck and finally prints them.

The redirection operators can be used in combination with each other.

5.8 Common Commands

kpasswd sets the AFS password for all unix computers.

tkdiff *file1 file2* shows the differences between file1 and file2.

links URL starts a text mode WWW browser. Ideal for browsing documentation. The URL argument is required.

ssh *user@hostname* logs in to the computer hostname as user user. This replaces telnet.

scp *user@hostname:remotepath localpath* copies a file from a remote computer to another computer on the network using the encrypted safe scp protocoll. This replaces ftp. user is the remote computers user name, hostname is the remote compters name, remotepath is the path to the file to copy on the remote computer, localpath is the path where the remote file should be copied to. Wildcards are supported.

sftp *user@computer* is a secure replacement for ftp. This can be used to transfer files back and forth between the computers involved. After logging in with your password, type help at the sftp prompt to get more information.

where *progname* lists each copy of the specified program in the current path. This is useful when it is unclear which version of a program is being used.

which *progname* prints the full path to the command progname.

who prints a list of users currently logged in to the system.

stty -a prints the current terminal characteristics.

stty rows *n* sets the teminal to n rows. Useful for Versaterm users.

stty columns *n* sets the terminal to n columns.

xpdf *file.pdf* is a pdf file viewer.

pdftops *file.pdf* converts a pdf file to a postscript file for printing.

ps2pdf *file.ps* converts a postscript file to a pdf file for exchange with your colleagues.

6 Editing under Unix

There is a choice of editors under unix. This section will first describe the more popular editors and then proceed to describe a specific editor **jed** in more detail. All editors can be started by typing their name or the name followed by a filename to edit.

ed a line oriented editor. Needs to get used to but is available on each unix system.

vi Another editor common on all unix systems. Difficult to use, but some love it.

emacs A very powerful and flexible editor especially recommended for programmers. The programming support is only surpassed by the VMS-lse editor. This editor comes with a built in tutorial which can be accessed by starting emacs (simply type *emacs*) and issuing the command *Ctrl-h t* within emacs. Emacs can be left with the command *Ctrl-x-c*.

nano,pico Tiny but sufficient editors with the same user interface.

joe A wordstar compatible editor. Help can be accessed by typing: *Ctrl-KH* after starting joe.

nedit A menu driven graphical editor very much like wordpad.

jed Is available both for text and graphical mode. As it has a nice edt emulation, it is described in more detail below.

6.1 Editing with jed (and emacs)

Another nice feature of jed is that it is available both in console mode and as a graphical editor. The console mode editor is started with the command *jed filename*. The graphical X-window version is started with *xjed filename*. jed is also known to work satisfactory through Versaterm on a Mac.

The following description of jed's working uses the default emacs mode. When arguments are required for commands jed will prompt for them in the bottom line. Some important jed key commands:

F10 access the menu. All commands are documented in there.

ESC leave menu and return to text.

Ctrl-X Ctrl-S save your work.

Ctrl-X Ctrl-F open a new file.

Ctrl-X i insert a file.

Ctrl-X Ctrl-C exits jed.

Ctrl-space starts a new marked region. Move the cursor to mark the desired region.

Ctrl-w remove the marked region.

Ctrl-y copy the last removed region into the buffer after the current cursor position.

Ctrl-X o move to other window.

Ctrl-X 1 Causes the current buffer to take up the whole screen.

Ctrl-X 2 split the screen.

Ctrl-S search forward.

Ctrl-% search and replace.

6.2 Editing with edt or xedt

On unix edt or the graphical version xedt are again jed but in the edt mode. Again, the most important commands:

F10 access the menu. All commands are documented in there.

ESC leave menu and return to text.

Ctrl-K Ctrl-W save your buffer with a new name.

Ctrl-ks saves your buffer.

Ctrl-K G open a new file.

Ctrl-K Ctrl-I insert a file.

Ctrl-K E exits jed.

Del starts a new marked region. Move the cursor to mark the desired region.

Num 9 remove the marked region.

Ctrl-y copy the last removed region into the buffer after the current cursor position.

Ctrl-W o move to other window.

Ctrl-W 1 Causes the current buffer to take up the whole screen.

Ctrl-W 2 split the screen.

Ctrl-F f search forward.

Ctrl-% search and replace.

7 Command Files or Shell Scripts

Quite often users wish to automate certain tasks. This can be done through command files or – in unix language – shell scripts. It turns out that unix has very powerful scripting facilities which come close to the possibilites of a full blown programming language. Writing a shell script basically involves three steps:

- Write the script
- Make it executable with the command **chmod +x scriptname**
- Run and debug it.

All normal unix comands and user programs can be executed from a shell script. Additionally the shell provides the usual programming constructs for loops and conditional execution. For more details, see the shell documentation *man csh* or borrow the shell programming book from Mark Könnecke, WHGA 112. As usual there are a few things noteworthy about shell scripts:

- As already stated there are several shells available under unix. As a default the system selects the simplest shell, the Bourne shell, for executing shell scripts. This might not be what you want. In order to have the shell script executed by the tcsh, commonly used interactively, add the line *#!/usr/bin/tcsh* as the first line, starting at column 1 to your shell script.
- Your shell script might be executed by other users or under circumstances where your login script has not been executed in advance. This means that environment variables and pathes you take for granted might not be available in your shell script. Thus it is always a good idea to set all required environment variables in your script and specify non standard programs and data files with the full pathname.
- Environment variables can be referenced with the **$variable** construct in your script.
- Command line parameters are available as **$1, $2, ..., $n** to your script.
- Shell scripts are executed in a separate process. This means that any environment variables set in your script are not passed back into the calling shell. Sometimes one might want to use a script for setting variables in your working shell, for instance in order to run a program which configures itself via environment variables. Such scripts have to be called with the construct *source scriptname* in order to achieve the desired goal.

7.1 The login script

When a user logs in interactively the shell sources a special script, the .tcshrc script, from the users home directory. This is the place where to put common environment variable settings and the like. If the .tcshrc file does not exist, create it with an edior. Two things are noteworthy about this .tcshrc script:

- In order to access all the NUM software on the central AFS–server the following line is required in the .tcshrc:

  ```
  source /afs/psi.ch/project/sinq/lns.login
  ```

- The tcsh supports an alias mechanism. An alias is an short name for a long definiton. The syntax is: **alias shortname longname** . For example the command *alias dir "ls -lisa".* defines a new command *dir* which can be called instead of *ls -lisa.* If an alias does not work right away, enclose the longname in quotes before running for help!

8 The AFS Filesystem

More information about AFS, its commands and usage can be found under:

http://ait.web.psi.ch/services/file/index.html

Each user has three special directories in her AFS home directory:

Backup This automatically contains your data as of yesterday. Use this to recover accidentally deleted files. Just copy the lost files with cp to wherever they should go.

private Files in this directory are private, i.e. not visible to anyone else in PSI (well, except AFS administrators of course).

public Files in this directory are visible and readable (but not modifieable) by any user. Use this to share data with your friends.

AFS file systems are under access control in order to prevent malicious users from destroying other peoples data. In order to do so, AFS uses Access Control Lists (ACL). Access control under AFS is per directory. There are a few commands for dealing with ACL's:

fs listacl for listing ACL's

fs setacl for setting ACL's.

The normal unix commands for controlling access rights, chmod and chgrp, do not work under AFS!

Each user has a disk quota on the AFS file system. Thus, if the error message: **Disk quota exceeded** appears either delete some files or apply for more space on AFS with a special form available online. Your quota and its usage can be queried with the command:

```
fs listquota
```

or

```
fs examine
```

AFS is designed to be used from a plethora of different operating systems. However, on each operating system different executables, libraries etc. are needed. In order to handle this AFS provides a neat facility: a special symbol @sys which can be used in pathnames. The @sys is translated by AFS into the system name. The usage of this can best be shown through an example: A user has two subdirectories in her AFS home directory:

i386_linux24 In this directory hierarchy all programs for linux are installed.

Now, in her .login the path string contains: /afs/psi.ch/user/h/hugo/@sys/bin. Then if the user logs in on an alpha computer, all alpha programs will be available, if the user logs in to a linux machine all linux programs will be available.

Other useful AFS commands are:

tokens shows you the expiry date of your login.

klog username asks for your password and then goes away and refreshes your login (or in AFS–speak, your token).

kpasswd changes the AFS password. This command only works properly when used on the PSI Linux cluster llc.

9 Frequently Occuring Problems

Here some common problems occuring under unix are discussed.

Error 1 *Applications running under an AFS account lock up for no apparent reason. Or no files can be written to a users home directory. Or things which used to work stop to do so. Programs start to behave funny.*

AFS password tokens are only valid for 25 hours! After that your login expires and nothing works anymore. Log out and in again to fix the issue. DO NOT complain, this is a security feature, no bug.

Error 2 *36736:/data/koenneck/bin/gimp: /sbin/loader: Fatal Error: Cannot map library libgtk-1.2.so*

Such an error message says that the system failed to find a shared library. Shared libraries contain system functions which are used by so many programs that they should be generally available without programs linking in the actual program code. In order to solve the problem these tow steps are required:

- Locate the library. It is the file ending with .so in the error message. Good places to look are /data/lnslib/lib and the /usr/local hierarchy. The unix utility *find* might help searching the library.

- set the environment variable LD_LIBRARY_PATH to point to the directory where the library was found. For example, the library for the above mentioned error message was found as */data/koenneck/lib/libgtk-1.2.so*. Then the command:

  ```
  setenv LD_LIBRARY_PATH /data/koenneck/lib:$LD_LIBRARY_PATH
  ```

  solves the prolem.

Error 3 *xv: Can't open display*

or

Error 4 *X11 connection rejected because of wrong authentication at Fri Jan 26 10:40:17 2001. a Rejected connection at Fri Jan 26 10:40:17 2001: X11 connection from lnsa15.psi.ch port 1590*

These errors happen if an X-windows program cannot connect to an X–server to do the graphics. To solve this follow these two steps:

- *Before* connecting to another workstation via ssh, issue the command **xhost +remotecomputer**. Replace remotecomputer with the name of the machine you are going to connect to. This step is usually not necessary on PC's. An example: **xhost +lnsa17** allows the computer lnsa17 to display on your workstation.
- On the remote computer, issue the command: **setenv DISPLAY yourcomputer:0.0** Replace yourcomputer with the name of the machine your are sitting in front off, i.e. at a workstation, the workstation name, at an X-terminal, the X-terminal name, at a PC, the PC name. An example: **setenv DISPLAY pc1530:0.0** tells X-windows to display all windows on the computer pc1530.

If this fails you may be sitting in front off a PC and not running the X11–application (Macintosh) or Reflection–X (PC). Running such programs are a necessary requirement for displaying graphical unix applications on a PC.

Error 5 *Interesting colour effects after starting a X–11 application*

When a X-server is running in 256 colour mode, there are exactly those 256 colours available for graphical applications. Some rogue applications reserve most of these colours for themselves. Then no more colour table entries are available for other programs. Such programs then make do with the available colours. The results are rarely pleasing. The solution is to close all other graphical X applications except the program you wish to run. Especially take care to close Netscape, it is a known rogue application. Sometimes SICS clients cause problems as well.

Some X programs cannot manage X servers with many colours. This mostly hits PC users. In such cases try various screen colour settings on your PC.

10 Programming under Unix

All examples in this section use Fortran as the example compiler. Most things apply in a identical manner to C, F90 or C++ programs as well, only the call to the compiler is different.

The following compilers are available:

- f77 Fortran 77 compiler
- gcc C-compiler
- g++ C++ compiler
- pgf90 Fortran 90 compiler. Please note, that this is a F90 compiler and does not support F95 constructs! Object files compiled with this compiler are not compatible with objct files compiled with the other compilers. This implies that a program using both f77 and f90 files must be compiled completely with pgf90.

10.1 Compiling

The command:

```
f77 -g -c program.f
```

compiles the F77 fortran program program.f to the object file program.o.

10.2 Compiling and Linking in one Go

Smaller programs can be compiled and linked with one command:

```
f77 -g -o program program.f
```

compiles program.f and links it yielding the executable program named *program*.

10.3 Linking

The command:

```
f77 -o myprog program.o
```

links the program and places the resulting executable program in the file myprog. All necessary object files and libraries must be given on the link line. The program can then be started by typing *myprog*.

10.4 Linking with Libraries

In larger projects parts of the code will typically reside in libraries. Unix has an automatic replacement scheme for library names. As an example take the library with the name xml. The corresponding library file is either libxml.a or libxml.so. This information is useful when trying to locate library files. Unix links against a library by specifying the path to the library with the -L option and the name of the library with the -l option. An example:

```
f77 -g -o myprog myprog.o -L/usr/lib -lxml
```

links against the library libxml.a or libxml.so residing in /usr/lib.

10.5 Building programs with Make

If a program consists of several files, it can become quite tedious to type all those compile and link commands. Of course this can be automated by shell scripts but the standard unix way of doing this is to use the make program. Using make has the advantage that only those files actually changed will be recompiled and linked. Moreover make is the standard way to build programs under unix. In order to use make a file named *Makefile* is created in the program directory which holds commands for compiling and linking the program. Then typing the command **make** is the only thing required to compile and link the program. Of course other filenames can be used for the makefile as well, but then make has to be invoked like this: **make -f mymakefile**. Writing makefiles can be complicated but a simple makefile is not to hard to do. Here is a simple makefile to use as an example for discussion. This example has line numbers added to it in front of each line which the actual makfile would not have.

```
01 #-----------------------------------------------------------------------------
02 # Makefile for focus simulation program
03 # Stefan Janssen, Joel Mesot
04 # ported to Unix & PGPLOT by Mark Koennecke, April 1998
05 #-----------------------------------------------------------------------------
06
07 OBJ = focus4.o plot.o
08
09 .f.o:
10  f77 -g -c $*.f
11
12 all: focus4
13
14 focus4: $(OBJ)
15  f77 -g -o focus4 $(OBJ) -L/afs/psi.ch/project/sinq/sl-linux/lib -lpgplot -lX11
16
17 clean:
18  rm *.o
19  rm focus4
```

Lines 01-05 constitute a comment.

Line 07 In this line a variable called OBJ is defined. This is a common one holding all the object files needed for the program. Lines can be continued on the next line when the current line is finished with a \. The construct $(OBJ) in line 14 references the variable.

Line 09 This line holds a suffix rule. This one tells make how a object file should be created from a fortran .f file. The string $*.f is replaced by make with the fortran file to compile.

Lines 14-15 is a so called dependeny line. Line 14 holds the name of an item we wish to build, in this case the program focus4. After the : there are the files needed for building this program. Not surprisingly these are our object files. In line 15, below

there is a command which tells make how to create focus4 from the objects. This is the link line. *Beware* lines of this type *MUST* start with a TAB character!

Line 12 By default make searches for a dependency called all in the file and executes everything required there. Which can be other dependencies.

Line 17-19 these lines define another target. This could be invoked by typing **make clean**. clean removes all object files and executables from the current directory.

The best way to write your own makefile is to use the one given above as a template and continue from there. More documentation about make is available both on paper and as a man page.

10.6 Debugging Programs

Usually programs do not work first time. Then it is VERY helpful to use an interactive debugger in order to locate the problem and eventually fix it. With an interactive debugger you may stop a program at well choosen points, single step it and view and modify program variables. The Linux debugger is started with *gdb programname*. Some important gdb commands:

help For online help

run arglist starts the program running

break file:line set a breakpoint at line in file. specified after run.

run runs the program within the debugger.

run arg arg ... arg runs the program. Command line parameters are specified after run.

n, next executes the next source line and stops. Jumps over functions.

s, step executes the next source line. If the line is a function, enter the function.

where print the current stack.

print var prints the value of a variable.

set var=val sets variable var to the new value val.

cont continue execution after a breakpoint.

quit exits the debugger.

All other commands are alike to dbx commands as described above.

10.6.1 Using core Files

When a program fails it usually creates a core dump. This is a special file, named **core.pid**, which contains some information about the program when it died. The pid part in the name is a number which was the unix process identifier of the broken program. The debugger can be used to look at this file. Invoke the debugger with:

```
gdb progname core.pid
```

A restricted number of debugger commands can now be executed. The most useful is **where** which shows you exactly at which line in which function the error occurred. Another useful command is **dump** or **dump .** which prints the values of the variables on the stack for you.

11 Unix in the SINQ Hall

In the SINQ hall unix computers are used for instrument control. This required some special setups which are described in this section.

11.1 Local Accounts

There are also a couple of local user accounts around, especially on the instrument computers. Local means that these user accounts are only valid on the computer where they have been defined. Typical examples for machines with local user accounts are Macintosh office systems and instrument computers.

11.1.1 Local Accounts on Instrument Computers

The expiration of the AFS login after 25 hours is the reason why there are local user accounts on the instrument computers. Only with these accounts continous operation of the instrument can be ensured. Moreover the instrument can be kept measuring even in the unlikely event of a network problem. Typically, on each instrument computer two accounts exist:

inst is the account under which the instrument control software runs. For example focus. Only instrument responsibles should access this.

instlnsg is a local account to for the use of guest scientists measuring at the instrument. For example focuslnsg on the FOCUS computer.

Please replace inst above through the name of the instrument.

11.1.2 Linux Clients in the SINQ Hall

In the SINQ hall, there are a lot of older Linux PC's which act as clients. These PC's can be used to run the client software needed to access the instrument control software SICS. Also, there are a couple of Linux PC's used for data analysis at the instrument. The linux client PC"s have a common local user account with the credentials: sinquser/sinquser.

12 Windows and Unix

This section describes how to escape the confines of Windows and use unix software from your Windows-PC.

12.1 Accessing Unix from Windows

The first thing to do is to login in to the unix machine. Two cases have to be distinguished: Text mode login or graphical login for using graphical X–windows applications.

12.1.1 Text Mode Login to Unix from Windows

A special terminal program is required in order to access unix computers from Windows. The recomended program is **putty** which should already be installed on your Windows–PC. Once started, the program will prompt you for a unix ssh-host, username and password. If this is done for the first time a few more questions are asked which can be safely answered with the defaults.

12.1.2 Graphical Login to use X–Windows from Windows–PC

In order to use graphical unix applications two programs need to be run on the PC: Reflection–X, which provides an X–server, and putty. For instruction how to set everything up, consult: http://ait.web.psi.ch/services/ssh/reflectionX-ssh.html. Sorry, this is involved but this is the price to pay for using Windows. The most recent version of Reflection–X supports ssh login directly.

12.2 Data Exchange between Windows and Unix

There are several possibilities to exchange data between Windows and Linux:

- With WinSCP
- With the Windows AFS–client.
- From Unix to Windows with the smbclient program.
- From Linux to Windows through ntmount.

WinSCP is a user friendly graphical client for the scp protocoll. Winscp is perhaps the easiest way of exchanging files between Windows and Linux. The interface is alike to the popular WinFTP program.

There is an AFS–client software installed on each PSI PC which can access data stored on AFS filesystems. When logged in to Windows–XP, click on the lock symbol, at the right, in the main windows toolbar. Under the tab labelled *tokens*, you can login to AFS, under the tab labelled *Drive Letters* you can assign windows drive letters to AFS–filesystem paths. Once this is done, data can be copied in normal windowish ways between AFS and your PC.

Another possibility is to use smbclient for connecting from unix to Windows–NT file servers. The syntax is:

```
smbclient share -U username
```

```
Example:
```

```
smbclient "\\scratch0\scratch" -U koennecke
```

The symbol share is the Windows–NT network path to the disk which should be accessed. username is your Windows–NT (e-mail) username. The program will then ask for your password and if all is well will connect you to the Windows–NT disk. The example, for instance, tries to access the PSI–scratch area on the NUM Windows–NT server scratch0 as user koennecke. Please note the quotes around the network path. Without them ugly things happen. After connecting, data can be retrieved using a syntax as used by many ftp command line clients. Type help in smbclient or see the man–page for smbclient. Please note, that only your network drives, (U:, G:, scratch areas) are accessible in this way. In order to access your local harddrive, a special setup on your PC is required.

Under Linux, there is another more comfortable way to access Windows user data. Type **ntmount**. After entering your password some Windows shared file systems will be mounted under ∼/nt/drive_letter. The shares to mount are configured in the file .ntshare in your home directory. At the bottom of this file there are entries of the form:

```
//lnsa15/koenneck       k
```

This means that the share koenneck on the server lnsa15 is to be mounted under ∼/nt/k.

13 Macintosh and Unix

13.1 Accessing Linux Computers from Macintosh

As MacOS X computers are actually unix computers, this is very easy:

- Start the X–server application, X11.
- In the X11 application type: **ssh -X user@hostname**. Enter your password and all is set.

13.2 Data Exchange between MacOS X and Unix

Files can easily be copied between MacOS X computers and unix through either the AFS–client software or through **scp** or **sftp** as described above for the general unix case.

13.3 File Exchange from MacOS X to Windows

By the way: Macintosh users can connect to Windows shares throug the finder application with the following address string:

```
smb://psi.ch;username@server/share
```

For example:

```
smb://psi.ch;//koennecke@scratch0/scratch
```

tries to connect to the central PSI scratch area as user koennecke. There is also a NUM group account on the NT–newtwork which can be accessed as:

`smb://psi.ch;//koennecke@psi23/3300`