

Introduction to *Unix* at DESY

There are only a few commands that you have to know in order to get started on Unix. These will be presented in this introduction. For a more detailed description of the syntax of the commands and the available options, you can consult the help files (so called manual pages) on the computer system or read a good Unix book from the DESY library (e.g. Mark Sobell, *A Practical Guide to Unix System V* or W. Abrahams and B.R. Larson, *Unix for the Impatient*).

Unix commands sometimes have cryptic names and a very strict calling sequence for their options and parameters. The commands are case sensitive, as are the file names on the system, but most of the Unix commands are all lower case letters. Options are usually preceded by a - (minus) sign, or sometimes by a + (plus) sign. The general syntax is:

command -options parameters

but of course there are exceptions. Most commands allow more than one option. These options can either be put individually on the command line (e.g. `cmd -1 -2 -3 ...`) or combined to a single option string (e.g. `cmd -123..`).

File and Directory Names

Unix has a tree like file and directory structure, where you can address any file or directory with its absolute path name starting at the file system root (denoted by /), or with a name relative to your current position in the file tree. Examples:

.	the dot denotes the current working directory
..	two dots denote the directory above the current one
/usr/local/bin/xrsh	is an absolute path name for the command file <code>xrsh</code>
<i>subdir</i>	is the name of a subdirectory below the current one
../ <i>otherdir</i>	is a directory parallel to the current one
~	your home directory
~ <i>username</i>	home directory of another user
. <i>controlfile</i>	name of a control file in the current directory (e.g. for mail, editors, etc)
	usually these "dot files" are found in the home directory

Manipulate Files

Wildcards (? single character, * word) are very useful in handling files. Be very careful when you use wildcards with the copy, move, or remove command. Unix will delete or overwrite existing files without warning.

<code>cat name</code>	list file with name <i>name</i>
<code>head name</code>	list first 10 lines of file (option <i>-number</i> , e.g. <code>head -20</code> for first 20 lines)
<code>tail name</code>	list last 10 lines of file (option <i>-number</i> , e.g. <code>tail -20</code> for last 20 lines)
	tail -f continuously lists the end of a file
<code>less name</code>	list file page by page (other pagers are more , pg)
<code>cp name newname</code>	copy file to new name or new directory
<code>mv name newname</code>	move file to new name or new directory
<code>rm name</code>	remove file (option <i>-i</i> will prompt you before deletion)
<code>file name</code>	analyze file type and contents
<code>find path-name expression</code>	find a file in the file tree (relatively slow)
<code>locate name</code>	locate a file in the file tree (fast, based on a periodically updated list)
<code>ln name newname</code>	link an existing file with name <i>name</i> to a new location/name <i>newname</i>

Manipulate Directories

mkdir *name* create a directory with name *name*
rmdir *name* remove the (empty) directory with name *name*
cd *name* change to working directory with name *name*
pwd print (=list) name of the current working directory
ls *name* list contents of the directory with the name *name*
 if *name* is omitted, the current working directory is listed.
 (options -l long detailed listing, -a lists also dot files)
 e.g. **ls -al** will list all files in the current directory in the long form

Editors

There is a large variety of editors on Unix. The basic editor **vi** is cryptic to learn but very powerful, while other editors, like **pico**, are very easy to learn but have only a limited number of commands. The editor that is recommended because of its widespread usage on all kinds of Unix platforms and VMS is *emacs*.

vi standard Unix editor (see USG/93/01)
pico an easy to use editor (see USG/93/02)
emacs the recommended editor (see emacs reference card and USG/93/07)
xedit an X based editor
ted a Motif style editor

Shells

The interaction between the user and the Unix system is controlled by a program called shell. There are two basic families of shells, the Bourne shell and the C shell family. High level shells are recommended for interactive work, while shell scripts should be written in a low level shell (sh). At DESY the following shells are supported:

sh basic Bourne shell
ksh Korn shell, which is a Bourne shell with additions
zsh Z shell, which is a Bourne shell with more additions
csh basic C shell
tcsh C shell with many additions

Printing

All PostScript printers at DESY are accessible from all computer platforms. The print request is spooled via central print servers which recognise various formats (e.g. text, PostScript, dvi, metafiles, etc.) and print them accordingly.

lp *name* print file (on System V Unix systems)
 options: -d*printer*, -n*number-of-copies*, -o*DESY-option*
 e.g. **lp -dr02ps3 -n5 -oc name**
 will print 5 copies of the file *name* on r02ps3 in a compressed format
lpr *name* print file (on BSD Unix systems)
 options: -P*printer*
 e.g. **lpr -Pr02ps3 filename** will print on r02ps3
lpdest list all printer destinations (DESY command)
lpq (lpstat) list jobs on the print queue submitted by lpr (lp)
lprm (cancel) remove job from the print queue submitted by lpr (lp)

Manipulate Variables

The behaviour of programs and of the shell is controlled by variables or environment variables. Variables in the Bourne shell family have upper case names while variables in the C shell family have lower case names. Variables are only valid for the current shell, while the values of environment variables are accessible on all subshells.

<code>echo \$VAR</code>	list value of variable <i>VAR</i> (remember: variable names are case sensitive)
<code>VAR=value</code>	set variable <i>VAR</i> (in Bourne shell family)
<code>set var=value</code>	set variable <i>var</i> (in C shell family)
<code>export VAR</code>	export value of variable <i>VAR</i> to the environment (in Bourne shell family)
<code>setenv var value</code>	export value of variable <i>var</i> to the environment (in C shell family)
<code>set</code>	list values of all local variables
<code>env</code>	list values of all environment variables

Command Execution

Commands can be executed in the foreground or the background. In foreground execution, input is read from the keyboard and output is written to the screen. If the command does not need input, the keyboard is locked until the command execution is finished. To avoid the keyboard locking, commands can be placed in the background for execution. This is achieved by appending an ampersand (&) to the end of the command. Examples:

`ls -al` will be executed in the foreground
`mosaic &` will be executed in the background

Whenever a command is executed, the system will start a process and assign a unique process id (pid) to it. For commands which are executed in the background, a job number will be assigned in addition. Background commands can be monitored on all but the Bourne shell with the following commands:

`jobs` get a list of all background commands, where the returned lines mean:
[job-number] process-id status command-name
`stop %n` stop job with job number *n*, can be restarted with `bg %n`
`kill %n` kill job with job number *n*
`notify %n` request notification from job *n* if status changes
`fg %n` move job *n* to the foreground and restart if necessary

All processes can be manipulated with the following commands:

`ps` list all processes started from your current window
`ps -ef` produce a full listing of all processes on the (System V) machine
`ps -aux` produce a full listing of all processes on the (BSD) machine
`kill pid` kill command with process id *pid*
`CTRL-Z` stop current foreground command
`bg` move current stopped foreground process to the background and restart

Note: background does not imply batch. All jobs for H1/ZEUS should be started using the NQS batch system.

Redirection and Piping

In Unix, the output of a command is usually written to the screen and the input usually given from the keyboard. Error messages are printed to a file called standard error (usually the screen). These standard file assignments can be redirected to or from any other file by the usage of a less-than-sign

(<) or a greater-than-sign (>) preceded by a 1 (optional) for standard output and 2 for standard error. On csh and tcsh the standard error cannot be redirected independently of the standard output.

```
cmd <file-in >file-out    will get its input from file-in and write its output to file file-out
cmd >>file-out            will append the output to an existing file file-out
cmd 1>file-out 2>error    will write output to a file with the name file-out
                           and error messages to a file with the name error
cmd >&file-out-err        will write output and error messages to a file with the name file-out-err
```

Instead of redirecting the output to a file, it could also be redirected to the input stream of another command. This is called piping and the piping symbol is a vertical bar (|). It is quite often used for listing the output page-by-page or executing search commands, e.g.

```
ps -ef | grep string    will look for a string in the list of all processes
news X11 | less         will list the news item about X11 page-by-page
```

How to Get Help

The standard way to get help on a Unix machine is with the so called manual pages. If the manual page for a command is installed on a system, you will get this manual page by typing:

```
man name-of-the-command
```

The disadvantage of the man command is that you need the exact spelling of the command name in order to read the corresponding manual page. If you don't know the name of the command you are looking for, some machines allow for an index-driven search for the manual page:

```
apropos keyword    will produce a list of manual pages containing this keyword
man -k keyword     will produce a list of manual pages containing this keyword
xman               an X-based manual browser tool (click on help-button to get help)
info               description for GNU products
xinfo              an X-based description for GNU products
insight            a Motif interface to the SGI documentation
```

Network Access

On the network there are two types of host: those who have a trusted relationship to each other and those who do not. The system administrator of a host will establish a trusted relationship only if all userids on the related hosts are unique. With the help of the .rhosts file you yourself can establish a trusted relationship between single users on single hosts. The .rhosts file should only be writeable by the user (i.e. the file should have the protection -rw-r--r--). The entries in this file consist of *hostname username* pairs.

```
telnet hostname    create an interactive connection to a remote host
rlogin hostname    create an interactive connection to a trusted host
3270 hostname      create an interactive connection to an IBM host
xrsh hostname      create a connection to a trusted host with full X access
ftp hostname       copy files from or to a remote host
rcp file1 file2     copy files from or to a trusted host, the filenames on other hosts can be specified
                   as userid@host:filename or host:filename
```

Miscellaneous Commands

<code>chmod <i>rights file</i></code>	change access rights (who±right) of files or directories, e.g. chmod g+x <i>filename</i> will add execution right for members of the same group
<code>passwd</code>	change your password
<code>who</code>	list all logged in users on local machine
<code>finger <i>user@host</i></code>	list information about users on local or remote machines
<code>last</code>	display last login information of users on the system
<code>grep <i>string file</i></code>	find a string or regular pattern in a file
<code>diff <i>file1 file2</i></code>	display differences between two text files
<code>cmp <i>file1 file2</i></code>	display differences between two binary files
<code>sort</code>	sort files in lexical/numerical order skipping fields/characters
<code>cut</code>	cut out columns, characters, etc from a file
<code>uc</code>	uncosyfy NEWLIB members which have been copied with ftp from the IBM
<code>df</code>	display free space on disk
<code>du [-k]</code>	display space used by directories and files [in kilobytes]
<code>tar</code>	tape file archiver (files with name <i>name.tar</i>)
<code>cpio</code>	copy file archives in and out
<code>(un)compress</code>	compress or uncompress files (files with name <i>name.Z</i>)
<code>which <i>name</i></code>	display the full path name for the command with name <i>name</i>
<code>wc</code>	count lines, words, and characters in a file
<code>talk <i>user@host</i></code>	talk to another user on a remote host

Text Formatting and Displaying

<code>latex</code>	run L ^A T _E X
<code>xdvi</code>	display dvi files on an X-terminal
<code>dvips</code>	create a PostScript file from a dvi file
<code>a2ps</code>	convert ASCII text to PostScript for printing (good for manual pages) the command will send the PostScript output directly to the printer e.g. man a2ps a2ps will print the manual pages for a2ps
<code>ghostview</code>	display PostScript files on an X-terminal

Electronic Mail and Information Services

There is a wide variety of mail interfaces on Unix. The standard **mail** or **mailx** are not recommended. Instead try using one of the programs mentioned in the following table:

<code>pine</code>	a simple e-mail program using the pico editor
<code>elm</code>	a wide spread electronic mail program
<code>mh</code>	another mail message handling system
<code>xmh</code>	an X based mail interface to mh
<code>mmh</code>	a Motif style interface to mh
<code>vm</code>	a mail interface for emacs

Important system messages are displayed when you log on to the system. This message-of-the-day can be reviewed with the command **less /etc/motd**. Other more long-term system news are kept in the local news system, while information which should be accessible on more than one computer is posted to one of the DESY newsgroups. These newsgroups are readable from **mosaic** and any of the many newsreaders, and to post messages you have to invoke one of the newsreaders listed below. None of the newsreader, however, is completely satisfactory. We recommended **tin** if you are a first time user or **mosaic** if you only want to read newsgroups.

less /etc/motd	display message-of-today
news	display system news
mosaic	a Motif style document browsing program, netnews reader, information source, phonebook interface, ... (see USG/93/06)
tin	a netnews interface for VT style terminals
xrn	an X based newsreader
mxrn	a Motif style newsreader
gnus	a netnews interface for emacs

Programming

C and FORTRAN programs can easily be maintained and run under Unix. To run a program you just type the name of the executable. The program is compiled and linked with the same command:

<i>cc progname</i>	C compiler, e.g. cc myprog.c compiles the C program myprog.c and creates an executable named a.out
<i>CC progname</i>	C++ compiler
<i>f77 progname</i>	FORTRAN 77 compiler, e.g. f77 -o myprog myprog.f -L/cern/pro/lib -lpacklib compiles and links a FORTRAN program myprog.f with the CERN packlib and creates an executable with the name myprog

Some compiler options are used frequently, others are machine dependent. In the next table some of the most common options are listed:

<i>-o name</i>	change the name of the executable from a.out to <i>name</i>
<i>-static</i>	allocate local variables statically and set value to zero
<i>-Olevel</i>	set the optimize level
<i>-g</i>	create additional symbol table for debugging
<i>-c</i>	just compile and produce an object file with the ending .o
<i>-lname</i>	use the library <i>libname.a</i> for linking
<i>-L dir</i>	directory where the <i>libname.a</i> libraries are kept (on HP only available for fort77 compiler, not for f77) the cernlibs are in /cern/pro/lib or /usr/local/cern/pro/lib the naglib is in /usr/local/naglib

Other options can be found by consulting the corresponding manual pages for the compiler.

If you have to maintain a larger program which consists of many subroutines and header files (some of which may be dependent on each other), you might want to put down the creation rules for the program executable in a file called makefile and keep your object codes in a private library.

make	maintain, update, and regenerate groups of programs see man make and USG documentation (USG/93/11)
<i>ar option library files</i>	archive object files in library, options are: t - table of contents of archive r - replace file in archive d - delete file from archive e.g. ar r libmylib.a sub.o will replace subroutine sub.o in library mylib
dbx	a debugger
gdb	a GNU debugger
cvd	a debugger with Motif interface

To use any of the debuggers your program needs to be compiled with the *-g* option. Note that this will enlarge the size of the executable.