Quick Reference

1 Log In Session

1.1 Log In

Enter username at login: prompt.
Enter password at password: prompt.

1.2 Change Password

passwd

1.3 Log Out

logout, exit or ctrl-D

2 File System

2.1 Create a File

cat > *file*	Enter text and end with ctrl-D
pico *file*	Edit *file* using the **pico** editor
joe *file*	Edit *file* using the **joe** editor

2.2 Make a Directory

mkdir *directory-name*

2.3 Display File Contents

cat *file*	display contents of *file*
more *file*	page through contents of *file*
less *file*	scroll through contents of *file*

2.4 Comparing Files

diff *file1 file2*	line by line comparison
cmp *file1 file2*	byte by byte comparison

2.5 Changing Access Modes

chmod *mode file1 file2* ...
chmod -R *mode dir* (changes all files in *dir*)

Mode Settings

u user (owner)
g group
o other (world)

+ add permission
- remove permission

r read
w write
x execute

2.6 List Files and Directories

ls	list contents of directory
ls –a	include files with "."
ls –l	list contents in long format

2.7 Move (or Rename) Files and Directories

mv *src-file dest-file*
mv *src-file dest-dir*
mv *src-dir dest-dir*

2.8 Copy Files

cp *src-file dest-file*
cp *src-file dest-dir*
cp -R *src-dir dest-dir*

2.9 Remove Files

rm *file*	remove a file
rmdir *dir*	remove an empty directory
rm -r *dir*	remove a directory and its contents
rm -i *file*	remove file, but prompt before deleting

2.10 Change Working Directory

cd	return to your login (home) directory
cd *dir*	change to directory *dir*

2.11 Find Name of Current Directory

pwd	display absolute path of working directory

2.12 Pathnames

simple:	One filename or directory name for accessing local file or directory. **Example:** foo.c
absolute:	List of directory names from root directory to desired file or directory name, each separated by /. **Example:** /src/shared
relative:	List of directory names from working directory to desired file or directory name, each separated by /. **Example:** Mail/inbox/23

2.14 Directory Abbreviations

~	Your home (login) directory
~username	Another user's home directory
.	Working (current) directory
..	Parent of working directory
../..	Parent of parent directory

3.0 Commands

3.1 Command line

command arg1 arg2 … argn

3.2 Wild Cards

?	single character wild card
*	Arbitrary number of characters
[abc]	single "a" , "b" or "c"
[m-n]	single character from the "m" to "n" interval

3.3 Redirection

command > file	direct output of *command* to *file* instead of standard output (screen), replacing current contents of *file*
command >> file	as above, except output is **appended** to the current contents of *file*
command < file	*command* receives input from *file* instead of standard input (keyboard)

3.4 Pipe

cmd1 | cmd2	"pipe" output of *cmd1* to input of *cmd2*

3.5 Command substitution

cmd1 \`*cmd2*\`	The output from *cmd2* is used an argument for *cmd1*.

3.6 Command lists

Command list is a sequence of programs.

cmd1; cmd2	Executes the *cmd1*, then the *cmd2*.
cmd1 && cmd2	*cmd2* is executed if, and only if, *cmd1* returns an exit status of zero.
cmd1 || cmd2	*cmd2* is executed if and only if, *cmd1* returns a non-zero exit status.

3.7 Subshell

(list)	The list is executed in a subshell.

4 Search Files

Search files:

find *path…expression*	recursively descends the directory hierarchy for seeking files that match with the expression

Search in files:

grep *string filelist*	show lines containing *string* in any file in *filelist*
grep -v *string filelist*	show lines **not** containing *string*
grep -i *string filelist*	show lines containing *string*, ignore

6 Timesavers

6.1 Aliases

alias *string command*	abbreviate *command* to *string*

6.2 History: Command Repetition

Commands may be recalled:

history	show command history
!*num*	repeat command with history number *num*
!*str*	repeat last command beginning with string *str*
!!	repeat entire last command line
!$	repeat last word of last command line

7 Process and Job Control

7.1 Important Terms

pid	Process IDentification number. See section 7.2.
job-id	Job identification number. See section 7.2.

7.2 Display Process and/or Job Ids

ps	report processes and pid numbers
ps gx	as above, but include "hidden" processes
jobs	report current jobs and job id numbers

7.3 Stop (Suspend) a Job

ctrl-Z	**NOTE**:process still exists!

7.4 Run a Job in the Background

Start job in background:
Add & to end of command.

Example: xdvi unixintro.dvi &

Force a running job into the background:

ctrl-Z	stop the job
bg	"push" the job into the background

7.5 Bring a Job to the Foreground

fg	bring a job to foreground
fg *%job-id*	foreground by *job-id* (see 7.2)

7.6 Kill a Process or Job

ctrl-C	kill foreground process
kill -KILL *pid#*	
kill -KILL *%job-id#*	
	see 7.2 for displaying *pids & job-ids*

Common UNIX Commands

This document presents a brief description of commonly used UNIX commands. The list is a small subset of the available commands and utilities. For more information on these commands and others not listed here, consult the online manual pages (see the man command).

alias *alias-term command-string*
The **alias** built-in shell command allows the entering of shorter or easy-to-remember names to execute longer or hard-to-remember commands. For example, entering **alias dir='ls -al'** will allow **ls -al** to be executed whenever the **dir** command is entered. Entering **alias** by itself will list all the aliases currently set for the user.

cat *file*
The **cat** command displays the contents of the file named by *file*. To display the file a screenful at a time, use the more command.

cd *directory*
The **cd** command moves you (changes your current working directory) to *directory*. Entering **cd** without the *directory* argument will move you to your home directory .

chgrp *groupname path*
The **chown** command changes the group of the file or directory, *path*, to group, *groupname*.

chmod *permissions path*
The **chmod** command changes the access permission associated with a file or directory ("file" will be used here to refer to either a file or a directory).

Each file has three types of access: read (r), write (w) and execute (x). In a **ls -al** file listing, the abbreviations appear in the columns on the left.

For files:

r	to see the contents.
w	to change the contents.
x	to execute

For directories:

r	to list the catalog.
w	to change a catalog entry.
x	to access

The access to a file can be controlled separately for three sets of users: the owner of the file (u), a limited group of users (g), and everyone on the system (o). In a **ls -al** file listing, the first three columns (starting in column two of the listing) are the r, w and x access allowed for the owner, the second three are the access allowed for the group and the third three are the access allowed for everyone else.

Permissions can be specified in numeric format or using the abbreviations above. For the numeric format, three numbers are specified where each number represents the access granted for one of the three sets of users. Each permission number is determined by adding up the value associated with each type of access: r = 4, w = 2 and x = 1. The numeric access specification is an absolute one; all three types of access for all three sets of users are reset according to the new *permissions*.
The *permissions* can also be specified using abbreviations rather than numbers. Using this method, some of the permissions can be changed without affecting others. The *permissions* format is **<u, g or o> <+ or -> <r, w or x>**. The **+** adds the access indicated to the file without affecting the other permissions. The **-** removes the access from the file.

chown *username path*
The **chown** command changes the ownership of the file or directory, *path*, to user, *username*.

cmp *file1 file2*
The **cmp** utility byte compares two files.

cp *file1 file2*
The **cp** command creates an identical copy of the file, *file1*, and names the copy, *file2*.

date
The **date** command displays the current date and time. Use **date -u** to see the time in Greenwich Mean Time (GMT), universal time.

diff *file1 file2*
The **diff** command compares the contents of two text files and displays the differences.

exit
The **exit** command terminates the current UNIX shell.

find *path expression*
The **find** command recursively descends the directory hierarchy for each path seeking files that match the expression.

finger *name@address*

The **finger** command displays information about user account with the specified address.

grep *pattern file*

The **grep** (**egrep**, **fgrep**) command searches one or more files, specified by *file*, for the text string specified by *pattern* (see Regular Expressions).

head *file*
tail *file*

The **head** and **tail** commands list the first (head) or last (tail) ten lines of your file. Including -n number-of-lines option may vary the number of lines listed. For example, **head -n 50 report1** will list the first fifty lines of the file *report1*.

kill *id-number*

The **kill** command terminates the process (see Process and Job Control) with the id, *id-number*. The process id can be determined with the **ps** command. Generally, the **kill** command is the last method tried to terminate a running program. e.g. **kill -9 18201** is a "sure kill" of process number 18201.

ln *source target*

The **ln** command makes a hard or a symbolic (with the option **–s**) link to the *source* file or directory with the name *target*.

ls *pattern*

The **ls** command lists the files and directories in a directory. If *pattern* is the name of a file, only that file is listed. If *pattern* is a directory name, the contents of that directory are listed. If *pattern* is omitted, all the files and directories in your current directory are listed. The output of the **ls** command may be piped into the more command to pause the listing after each screenful of text (e.g. **ls -al | more**).

Here are a few of the options for the **ls** command (for a complete list, see the ls man page):

-a

lists *all* files in the current directory. Without this option, filenames that begin with a period (such as .bashrc, .login, and ..) are not shown).

-l

lists the filenames in *long* format. This format includes the protections (changable with chmod) on each file and the owner of the file.

man *command*

The **man** command displays the standard UNIX manual page for the *command* you specify.

mkdir *dir1*

The **mkdir** command creates the directory, *dir1*, within your current directory (unless the specification of *dir1* begins with a /).

more *file1*

The **more** command displays the contents of the text file, *file1* , a screenful at a time, pausing at the end of each screen until the user presses one of a few special keys.
more may also be used at the end of a "pipe" to cause the output from another command to be paused a screen at a time.

mv *name1 name2*

The **mv** command moves and/or renames the file or directory, *name1*.

passwd

The **passwd** command changes your UNIX login password. After entering the command **passwd**, you will be asked to enter your old password, then the new password that you want to change to, and then the same new password again.

ps

The **ps** command displays a list of the processes currently running on the machine that you are logged into. If no arguments are entered with the **ps** command, only the "important" processes, that you own (i.e. that you are running) are displayed. The **-a** option includes processes owned by others in the list. The **-g** option includes all processes, not just "important" processes. The **-u** option provides more information for each processes than is printed by default. To terminate a process, see the kill command.

pwd

The **pwd** command displays the full path of your current working directory.

rm *yourfile*

The **rm** command removes the file, *yourfile*, permanently from the filesystem.

rmdir *dir1*

The **rmdir** command deletes the empty subdirectory, *dir1*. To delete non-empty subdirectories, see rm -r.

sort *file*

The **sort** command sorts lines of all the named files together and writes the result on the standard output.

su *username*

The **su** command allows a user to assume the identity and permissions of another user, *username* (provided that the password for *username* is known). The **su** session is ending by entering the command, exit.

tail *file*

See head.

wc *file*

The **wc** command counts the number of words or characters and lines in your file. If the **-l** option is used, only the number of lines is counted.

who

The **who** command displays a list of who is logged on to the system and where they are logged on from. See also finger .

Regular Expressions

c	non-special character *c*
\*c*	special character *c*
^	beginning of line
$	end of line
.	any single character
[*abc*]	any character *a*, *b*, or *c*
[*a–c*]	any character in range *a* through *c*
[^*abc*]	any character except *a*, *b*, or *c*
[^*a–c*]	any character except characters in *a–c*
\*n*	*nth* \\(...\\) match (**grep** only)
r*	zero or more occurrences of *r*
r+	one or more occurrences of *rexp*
r?	zero or one occurrence of *r*
r1* | *r2	regular expressions *r1* or *r2*
\\(*r*\\)	tagged regular expression *r* (**grep**)
(*r*)	regular expression *r* (**egrep**)

Bourne Shell

Running Scripts

sh file [arguments]
or if the file is executable (set x bit), then
file [arguments]

Special Characters

* ? [ ]	< > | &	$ { } ; ( )
‘ ‘	“ “	\` \`

Shell Variables

var1=value1
var2=value2
echo $var1 ${var2}xx

Special Variables

$*	all parameters
$@	same as the **$*** without “”. Example: “$*” == “$1 $2…” “$@” == “$1” “$2”…
$#	the number of parameters in decimal
$?	the status of the most recently executed foreground pipeline
$-	the current option flags
$$	the process ID of the shell
$!	the process ID of the most recently executed background (asynchronous) command
$0	the name of the shell or shell script
$IFS	the **I**nternal **F**ield **S**eparator that is used for word splitting after expansion
$HOME	home directory of the current user
$PATH	search path for commands
$CDPATH	search path for the cd command
$PS1	value of this parameter is expanded and used as the primary prompt string
$PS2	value of this parameter is expanded as with PS1 and used as the secondary prompt string

Parameter Expansion

${parameter-word}

Use Default Values. If parameter is unset or null, the expansion of word is substituted. Otherwise, the value of parameter is substituted.

${parameter=word}

Assign Default Values. If parameter is unset or null, the expansion of word is assigned to parameter. The value of parameter is then substituted.

${parameter?word}

Display error if null or unset. If parameter is null or unset, the expansion of word is written to the standard error and the shell, if it is not interactive, exits. Otherwise, the value of parameter is substituted.

${parameter+word}

Use alternate value. If parameter is null or unset, nothing is substituted; otherwise the expansion of word is substituted.

Shell Builtin Commands

:	no effect
. filename	include
break [n]	exit from a loop
cd [arg]	change directory
continue [n]	resume the next iteration of the loop
echo [arg]	arguments are written to the standard output
eval [arg]	the arguments are read as input and the resulting command(s) executed
exec [arg]	the command is executed in place of this shell without creating a new process
exit [n]	exit with the exit status specified by n
export [var]	export variables
hash[-r][name]	recalculation of the hash table
pwd	the current working directory
read var …	one line is read from the standard input
readonly [var]	the given variables are marked readonly
return [n]	causes a function to exit with code n
set [opt[arg]]	set flags
shift [n]	parameters from $n+1... are renamed $1...
test	evaluate conditional expressions
times	print the accumulated user and system times for processes run from the shell
trap [arg[n]]	signal handling
type [name]	indicate how it would be interpreted if used as a command name
ulimit [opt][n[	prints or sets hard or soft resource limits
umask [nnn]	user file-creation mask is set to nnn
unset [var]	remove the corresponding variable
wait [n]	wait for your background process whose process id is n and report its termination status

Evaluate Conditions (test)

The test utility evaluates the condition and indicates the result of the evaluation by its exit status. An exit status of zero indicates that the condition evaluated as true and an exit status of 1 indicates that the condition evaluated as false.

In the second form of the utility, which uses **[ ]** rather than test, the square brackets must be separate arguments and condition is optional.

-r filename	True if filename exists and is readable.
-w filename	True if filename exists and is writable.
-x filename	True if filename exists and is executable.
-f filename	True if filename exists and is a regular file.
-d filename	True if filename exists and is a directory.
-s filename	True if filename exists and has a size greater than zero.
-t [fildes]	True if the open file whose file descriptor number is fildes (1 by default) is associated with a terminal device.
-z s1	True if the length of string s1 is zero.
-n s1	True if the length of the string s1 is nonzero.
s1 = s2	True if strings s1 and s2 are identical.
s1 != s2	True if strings s1 and s2 are not identical.
s1	True if s1 is not the null string.
n1 -eq n2	True if the integers n1 and n2 are algebraically equal. Any of the comparisons **-ne**, **-gt**, **-ge**, **-lt**, and **-le** may be used in place of **-eq**.

These primaries may be combined with the following operators:

!	Unary negation operator.
-a	Binary and operator.
-o	Binary or operator (-a has higher precedence than -o).

Conditionals in Shell Scripts

If statement

```
if cmds
    then cmds
    else cmds
fi
```

Case statement

```
case word
    string1) cmds ;;
    string2) cmds ;;
    *) cmds ;;
esac
```

While loop

```
while cmds
    do cmds
done
```

Until loop

```
until cmds
    do cmds
done
```

For loop

```
for i in w1 w2 …
    do cmds
done
```