

Intro to Unix
Edited from A. Vazquez's tutorial document.

Additional resources:
Kelly-Bootle: Understanding Unix, Sybex Inc. 1992
<http://www.sph.umich.edu/~nicholst/UnixBrownBag.pdf>
+... a wealth of other sources.

Introduction

Quick Historical Perspective

Developed by Bell Labs (AT&T) around 1970
"Scientific" environment with the goal of being extremely robust
Different versions of UNIX evolved usually adapted to particular platforms (HP, Xerox, Linux,...)

What is UNIX?

Operating System: specifies the low-level use of the computer including hardware and software

Why UNIX?

Comparison with other operating systems:

DOS - similar to UNIX because of its command line approach, everything else
MacOS: similar to Microsoft Windows but more robust
Windows/WindowsNT/newer MacOS: operating systems which are converging towards
UNIX but are not quite there yet (and not in the foreseeable future either.
Linux - Adaptation of Unix for the Intel architecture by Linus Torvals.

Advantage of UNIX over these operating systems

Multi-tasking: running more than one action/program (process) simultaneously
(DOS, MacOS you can not use 2 applications at once or copy 2 separate files at once)

Multi-user: securely handle applications, files and processes from other users
DOS has no multi-user capability, MacOS has pseudo-multi-user capability
since allows different users, but the security of the files is not very robust

Multi-platform: the same OS implemented on different platforms such that the same
program run seamlessly across platforms
DOS for the Mac? There is Windows for the Mac but it is an emulator not an
operating system
UNIX is not truly multi-platform but the availability of true UNIX-like
operating systems for other platforms make the transportability of applications
easier.
The implication of this is that programs that do not depend on certain low-level
system/hardware calls will just need to be re-compiled

Network robustness: accessibility of the computer and its resources in a very controlled
fashion over the network
automount/export of file systems and directories
remote login and remote execution of processes
Note:UNIX recognizes the difference between remote and console users,
console having specific benefits over remote users

The UNIX Shell

What is the UNIX shell?

At the heart of Unix is a program called the “kernel”. It executes all the commands.

The shell is the intermediary between the user and the operating system (kernel) it serves as a "command interpreter"

Possesses very important functionality: environment variables, ability to execute functions and store variable values, ...

It allows the user to customize, program a session

There are a variety of shells with different functionality, such as, c-shell (csh), tc-shell (tcsh), bash-shell (bsh), korn-shell (ksh), z-shell (zsh), ...

Examples of differences: file completion, history, command line navigation, scripting syntax, handling of environment variables, ...

Most system scripts are written using the Bourne shell (sh), however we are only going to concentrate on the csh and/or tcsh shells

Use the following command to invoke the shell

```
tcsh          or          /bin/tcsh
```

To get out of the shell, type `exit` (then another exit or logout to logout)

To know which shell you are using type: `echo $SHELL` or `echo $shell`

Windows environments are usually a compilation of command line commands that get executed without the user noticing

Every time a new shell is called or executed, the user's shell resource file is read (e.g. `.cshrc`). If you want to re-call or force a call to a resource file, use: `source`

Overview and Getting Started:

UNIX is a multi-user operating system with excellent security

Implication is that there must be a user capable of establishing these securities/ownership. The name if this almighty user is root.

In most occasions each user has control over everything owned by the user along with root

The operating system recognizes two essential things for each user: username and password (root is not an exception)

Logging in:

```
telnet login.itd.umich.edu
```

```
ssh login.itd.umich.edu
```

```
or          ssh <hostname> -l <user>
```

username: enter username <enter>
password: enter password <enter>

Logging out:

logout

exit

Changing your password,

passwd

When you login, what happens?

Your username and password are verified

The entry for the user's shell choice is executed (the resource file for that shell is executed and if you are in the console other files also get executed to run your selected window manager)

Execution of .cshrc, or .bashrc ...etc.

Moving Around the Command Line and Some Sample Commands:

The typical unix syntax for a command is:

command [-options] [arguments]

Some commands:

who outputs who is logged in the computer

write <username> <terminal> writes <message> to <username> in <terminal>

<message> <ctrl-d> screen (e.g. pts/0), to end write use <ctrl-d>

hostname outputs the name of the computer

who am i outputs the detailed version of who

date outputs the date and time

finger -l outputs the details of the user being requested

passwd prompts for the change of password of the logged user

<ctrl-a> move to the beginning of the line

<ctrl-e> move to the end of the line

<ctrl-k> erase everything forward from the prompt

<ctrl-u> erases the entire line

<ctrl-d> marks the end of input in a line

We can use ; to execute more than one command in the command-line

Getting Help in the Command Line:

When you have a command and do not know how to use it, usually just typing the command utputs the instructions on its usage, or typing the command followed by "-h"

If no usage contents show up, you can read the manual pages on that command by typing "man" followed by the command of interest

```
man who
```

```
man ls
```

What if you don't know a command? You can search the man pages for keywords

```
man -k copy
```

```
apropos move
```

Another place you may be able to find help (if your operating system is Solaris) is in the Sun documentation web site (<http://docs.sun.com>) or anywhere in the web. The support for UNIX in the web is outstanding!

The UNIX File System

Each unit of storage belonging to a common collective is called a file

Files are organized in a specific fashion determined by the file system. Files can be written in text (ASCII), binary or both

The file system organizes the individual files in directories. Directories are organized by the file system in a tree-like fashion, where the basic or initial directory is the root directory

The file system is responsible for time-stamping the creation of files, setting the rights for ownership, size of the file, ...

Note that a strength of UNIX is that the file system is not determined by the operating system! This implies that you can choose which file system better suits your network. We are ONLY going to concern ourselves with NFS. Other file systems are AFS (used mostly for large networks), ...

Disks and Volumes/removable media

There are several types of file systems that UNIX (and most operating systems) recognize: fixed disks (i.e. hard drives) and removable media (i.e. cd-rom, floppy disks, optical drives, ...)

Even though the file system in these two is the same, the way the system handles these is not

In some UNIX systems (e.g. Solaris) removable media are called volumes

```
df          or          df -v
```

```
/ /dev/dsk/c0t0d0 384871 245577 100807 71%
/proc /proc 0 0 0 0%
/dev/fd fd 0 0 0 0%
/var /dev/dsk/c0t1d0 384871 177667 168717 52%
/opt /dev/dsk/c0t0d0 384871 10427 335957 4%
/tmp /dev/dsk/c0t1d0 96031 146 86282 1%
/usr/dt /dev/dsk/c0t0d0 96031 62586 23842 73%
/usr/openw /dev/dsk/c0t1d0 384871 195246 151138 57%
/var/cache /dev/md/dsk/d10 189767 23627 147164 14%
df: cannot statvfs /var/log/local: Permission denied
/ticket swap 1024 154 870 16%
/afs AFS 9000000 0 9000000 0%
```

The command `df` displays the capacity and usage of the file systems "mounted" on the host computer. The first 2 columns display the device, followed by its capacity, amount used, amount available, and capacity percentage (% full). We should always pay attention to the following areas: `/`, `swap`, `tmp`. These have an effect on performance.

If you insert a floppy into the floppy drive in your system and you go to `/floppy` and your system does not recognize the presence of the floppy you can tell the system to check the volumes by typing

```
volcheck
```

To eject the floppy

```
eject floppy
```

File System Organization

The collective of characters (bytes) that make up a file are identified by the filename

Files are placed in directories or folders, directories are also denoted by directory names

After logging in a user is placed in a particular place of the file system by default: the home directory. The home directory is the basic organizational area of a user. The location of the home directory is determined by root, however it is usually `/home` (Note that root's home directory is `/`)

In UNIX terminology the current directory is called the working directory, so when you log in your working directory is the home directory.

To find out the location of your home directory or where you are in the file system structure at any point, type

```
pwd
```

The file system organizes all directories under the root directory (/), so that / is the most basic directory, it has no "inheritors".

The owner of the root directory is the root account (root).

Let's explore /

```
cd /
```

```
pwd
```

The change directory command or cd allows you to move around the file system structure. To move into a location that is several directories "deep" separate the directory names by slashes (slashes denote directories)

To see where are you located in the file system structure use the present current directory command or pwd.

```
ls
```

To observe the contents of a directory use the list command or ls

Note that several directories should be present, for example, bin (programs), usr (user related), etc (system related), tmp (temporary – very useful directory), ...

Now let's go back to our original directory, the home directory, type

```
cd ~      or      cd ~<username>
```

```
pwd
```

In UNIX lingo (~) means user. The (~) followed by a space means "me-user"

Now, let's explore a directory and the files more in-depth, type

```
ls -l
```

```
ls -al
```

When a command in UNIX is followed by arguments that start with (-) these are usually referred to flags or options. In the ls command, the a option means show hidden files (files starting with a (.) dot), and the l options means show in long form (all the details)

Sample output of ls -al

```
total 86
```

```

drwxr-xr-x  8   towi  root  2048  May 17 10:48  .
drwxr-xr-x  3   bin   root 10240  Sep 21 14:02  ..
drwxr-xr-x  2   towi  root  2048  Aug 11 1992  .AppleDouble
drwxr-xr-x  2   towi  root  2048  Jul 10 1992  .netscape

```

total: indicates the amount of storage used by the files in this directory
type: d (directory), - (file), l (link) , also called file system units
permissions: rwxrwxrwx = read, write, execute permissions of owner (you in this case), your group and everyone else
Note that fields that are NOT permissible are marked by (-)
links: the total number of files and directories linked to the file system unit
owner: the name of the user who owns the file system unit
group: the group that owns the file system unit
size: file size in bytes
date: date and time the file was last modified (not the creation date)
Note: that if the modification date exceeds a year, the full date is shown, no time

Hidden files, files which will not be listed in an ordinary ls call, begin with a (.)

The dot-directory (./) indicates "this" directory

Note: the command cd . will not change the working directory

The dot-dot-directory (../) indicates the "previous" directory or parent directory

Note: the command cd .. will move you one directory up from the [previous] working directory and the command cd ../../ will move you up 2 parent directories

The permissions field is tricky. You can imagine this field being composed of 9 bits (binary units), which correspond to rwxrwxrwx in that order. The first 3 bits control the owner, the following 3 the group and the last 3 the public in general

Implications:

- Directories which are not marked readable can not be read by appropriate users
- Directories which are not writeable can not be used to store files
- Directories which are not executable can not be accessed by appropriate users
- Files which are marked readable can be viewed by appropriate users
- Files which not writeable can not be modified by appropriate users
- Files which are not executable do not allow the use of program files by appropriate users
- Only the owner of a file (or directory) is allowed to change the permissions

Also: `ls -F`

The -F option in the ls command shows you which file system units are directories, executable files, plain files and links through the use of slashes (/), stars (*), spaces () and ampersands (&), respectively.

Changing permissions to a directory

```

mkdir <directory_name>
ls -l <directory_name>
chmod 777 <directory_name>
cd <directory_name>
pwd
ls -l
cd ..
pwd
ls -l
chmod 666 <directory_name>
cd <directory_name>
chmod 755 <directory_name>
cd <directory_name>
rmdir <directory_name>

```

The command `chmod` changes the permissions of files, directories or links. The question now should be, what is 755? If we build a binary table we could see the meaning of 755, here are the results of such a table

7	read,write,execute ($1*2^2 + 1*2^1 + 1*2^0 = 7$)
6	read,write
5	read,execute
4	read
3	write,execute
2	write
1	execute
0	none

The first digit controls the owner's permissions, the second the group's permissions and the last digit the public's permissions

For example: If you want the directory to only be readable by you use `chmod 700 <directory_name>`

To change the ownership of a file or directory use,

```
chown <newuser> <file(s)>
```

We may want to use `-v` or `-c` flags to see exactly how the change in ownership takes place. See the man pages for more information. Note: only owners can change ownership.

We used the commands `mkdir` and `rmdir` to create (make) and remove directories. Directories with contents can not be removed, the contents must be removed first

To copy and remove files we use `cp` and `rm`, respectively. Caution should be taken with these commands in case of overwriting a file with the same name and removing a file that is not intended to be removed!

Note: UNIX file systems do not allow the recovery of files that are removed (back-up!)

The `-i` flag in `cp` and `rm` requests the system to prompt you whenever you attempt to remove or overwrite a file

`cp` and `rm` can be used to copy directories and all their contents by using the `-r` (recursive) flag

When a file is copied the permissions are also copied

Note: if a file is readable by you, it can be copied

```
cp <filename> <newfilename>
ls -l
cp -i <newfilename> <filename>
ls -l
rm -i newfilename
ls -l
cp <filename> <directory_path>/<newfilename>
cp <filename> <newfilename>
rm <newfilename>
cp -r <directory_path>/<directory_name> .
rm -r <directory_name>
```

To rename or move your files use the mv command. If no directory is provided the file is going to be renamed, however if a different directory is provided the directory is going to be moved.

```
mv <filename> <newfilename>
mv <filename> <directory_path>/.
mv <filename> <directory_path>/<newfilename>
```

Looking at Your Files

If the contents of your file are text characters, there are a variety of commands to inspect its contents: more, cat, head, tail, ...

```
cat <filename>

head <filename>

head -10 <filename>

tail <filename>

tail -40 <filename>

more <filename>
```

The command cat displays the entire text file, while head displays the text contents in the first n lines (20 is default) and tail displays the contents in the last n lines of the text file

The command more is has better functionality. It displays a screen-full of lines and awaits a key press to display the next screen-full of lines. Some tricks:

<space>	next screen-full of lines
<return>	next line
/<string>	to find the string in the file (forward search)
?<string>	to find the string in the file (backward search)

q

to exit more

For ascii and binay files use `strings` or `od` to view an interpretation of the file contents.

Wildcards: When a number of files are names in series common portions of the file can be used to specify the conglomerate of files with the use of wildcards.

Wildcards are special characters: * , [.] , ?

How to use these?

```
ls mr*           (anything that starts with mr)
ls *test*        (anything that contains test)
ls s11.00[1-9]   (files names s11.001 thru s11.009)
ls s11.0?1       (only the 6th character doesn't matter)
```

Wildcards can be used whenever a filename is to be denoted. For example, the can be used in `cp`, `mv`, `rm`, Not all commands allow the use of wilcards.

```
cp s11.0* /tmp/.
```

Finding Files:

To find files use the following command (a bit bizarre!)

```
find where_to_start_searching [-type <filetype>] -name
"<filename>" -what_to_do_with_the_result
```

```
find . -type f -name "<filename>" -print
```

```
find . -type f -mtime -2 -print
```

The above command searches for the filename in current and beyond, then prints the results in the screen. The following command has the `-mtime` flag which searches for the modify time, the following flag `(-2)` indicates the number of days. Note that wildcards may be used.

```
locate <string>
```

The `locate` command (may not be available in all systems) looks for files or directories that contain `<string>` in their name

Links:

Space is commonly a problem in file systems, wouldn't we know...

To avoid having duplicates of the same files in the same file system (or across file systems that are properly mounted/exported) we can create links or pointers to these files or directories

The `ln` command: `ln -s <file_or_dir> <linkname>`

```
ln -s ~/towi/Public/towi_data/readme.txt mylink1

ls -l

cat mylink1

pwd
```

Links can be a very powerful tool to better organize your files and save disk space!

We should always use the `-s` flag (soft links)

More File System Commands

The command `ls -l` tells us how much space individual files take up, but notice it does not tell us how much space directories use (it only tells us how much space the directory name/id takes up). To know how much space the directories and files in your current path are taking up, type

```
du -sk .
```

To know how much space your current directory is consuming, use

```
du -sk *
```

Typically the output of `du` is defaulted to block and not bytes (1 block = 1024 bytes). We use the `-sk` flag to show the summary of the output and the sizes in kilobytes.

Another very helpful command is `tar`. We use it to make a single file that contains a collection of files of files and directories (easy to store and ftp).

```
tar cvf <tarfilename> <files_and/or_directory_list>

tar xvf <tarfilename>
```

The `c` flag indicates "create tar file" and the `x` flag indicates "extract tar file". So, the top command creates a tar file with name `<tarfilename>` and it contains all files and directories which follow `<tarfilename>`. The following command extracts the contents of `<tarfilename>` in the current directory. We can tar files by providing all the files (with their path/location), but its easiest use is to make a directory organized with the files we want to tar. Here is a sample

```
tar xvf mytar1.tar images           to create the tar file

tar cvf mytar1.tar                 to extract the tar file

tar tf mytar1.tar                   to list the contents of mytar1.tar
```

Some of the tar options (flags):

```
c      create
x      extract
v      verbose (show me what you are doing)
f      file (tar to file, we will almost always use this option)
```

h follow symbolic links
l tell me if symbolic links can not be followed
i ignore problems, continue tarring

CAUTION: Usually tar files are huge!

A way to solve this is to compress the tar file. To compress files we can use the compress command (files end in .Z) or gzip (files end in .gz). To uncompress we use uncompress or gunzip. The difference between these is the compression routine, gzip is more effective than compress, so try to use gzip.

```
gzip <filename>
```

```
gunzip <gzippedfilename>
```

Some flags of interest are -v (when compressing) to know the effectiveness of the compression, and the -l flag (for compressed files) to know the compression stats.

The UNIX Command-line More in-depth:

All the commands we type in the shell command line are logged for us. The tcsh shell allows us to recall these commands!

```
history  
!  
!!
```

History displays the list of the logged commands that have been executed in your shell. We can recall old commands by typing ! (bang) and the command history #. If we simply want to execute the last command we entered, use !! (bang-bang).

Another way to re-call previously entered commands is to type a portion of command you want to re-call and press <esc-p>.

Redirecting input and output:

Other quick tricks of interest are the re-directs, which are | , < , << , > , >> .

Pipe (|) directs the standard output of one command into the standard input of another command.

```
ls -l | more
```

This command will take the output of ls -l and show it to you in more fashion.

Pipe becomes very useful when you want to find text.

```
grep <text> <filelist>
```

```
ls -l | grep Aug or ls -l | grep "Aug"
```

Grep looks in <filelist> for <text> and outputs the lines where <text> appears. Note that the input of grep are files. So, we can change its use slightly by having the input of grep be the output of other commands. In the bottom command, only the ls items which contain Aug will be listed. Similarly, we can use `du -a | grep <filename>` to find a file!

The other re-directs are useful also. Re-direct in (<) and re-direct out (>) work like arrows. They can take the contents of a file and push them into another command or out to another file.

```
ls -l > myls1.txt  
ls -l / >> myls1.txt  
cat myls1.txt | tail -2 > mylsend.txt  
vi file1.txt < stuff1
```

The first command takes the output of `ls -l` and puts it in the file `mysl1.txt`. The second command takes the long listing of the root directory and appends it to the file `mysl1.txt`. If the re-direct was not doubled (>>) the operation would be invalid since the file `mysl1.txt` already exists. So, the double re-direct indicates append if necessary. The last command takes the contents of `stuff1` and puts them in `file1.txt` via the editor (this might not work, only for presentation purposes).

Parsing text with the `cut` command. This command can be used to select the nth field in a line of text. For example, if a text file contains four columns of data you can select out just the second one by

```
cut -d ' ' -f2 filename
```

Where,

-d defines the field separators. This could blank space, commas, colons ...etc.
-f determines which field you want (note that there is no space after -f)

This is very powerful if you want to extract key parts of text when used in combination with `grep` and different pipes.

As we saw earlier links can be very useful. We can also make links to commands, called aliases (do not confuse this with Mac's aliases which are links in UNIX). Here is how alias works,

```
alias <newcommandname> "<command>"  
unalias <newcommandname>
```

The `alias` command links the <newcommandname> with <command>. `Unalias` undoes the alias.

```
alias ls "ls -F"  
alias h "history"  
alias rm "rm -I"
```

```
unalias rm
```

```
alias qix "rlogin qix.itd.umich.edu"
```

Now, when we type `ls`, it is going to be like typing `ls -F`. When we type `qix`, it is going to establish a remote connection to `qix`.

When files are executable such as scripts or programs, sometimes there are several versions of the same (or similar) programs. How do we know which one is being executed?

```
which matlab
```

```
which tcsh
```

These commands will return the full path of the executable file being used. If the output of this command is "no <filename> ..." then you need to specify the operating system where to find this command. This can be done in two ways: write the full path and command in the command-line or add the path and rehash.

```
/usr/bin/tcsh
```

```
or
```

```
setenv $PATH = $PATH:<fullpath_to_add>
```

```
rehash
```

```
<executablefilename>
```

You may want to use `echo $PATH` to check that the directory you want to add is not already in your path. The command `rehash` tells the computer to look again at your path and find the executables.

UNIX Processes:

Sometimes we run commands (like `gzip`) that take a really long time to finish and hold the command line. Since UNIX is a multi-tasking operating system, we can instruct the system that we want to run more than one command at once. We do this by putting the current command on background. This can be done two ways:

```
gzip mytar1.tar &
```

```
or
```

```
gzip mytar1.tar
```

```
<ctrl-z>
```

```
bg
```

The ampersand (&) instructs the system to run the command in the background. If the current process is taking a while, we can press <ctrl-z> (suspend current process), and then enter `bg` to indicate that the current suspended process will be put on background. Similarly, we can use `fg` to put the process in foreground and hold the command line.

The command `jobs` outputs the processes which are currently running. If you want to bring a job to foreground, enter `fg %<job#>`. If we want to cancel this job press <ctrl-c> (since it is in the foreground).

We can also check the processes we are currently running using,

```
ps
PID  TT TIME COMMAND
2916 pts/0 0:00 -tcsh
```

The output of `ps` is different than `jobs`. Here the processes are identified by the process-id (PID), the terminal number from which the process was started (TT or TTY), the time (in minutes and seconds) the process has been running and the command. You may want to use the flag `-e` or `-A` to have `ps` display ALL the processes running in that machine. For example,

```
ps -el      or      ps -elf
ps -uga
```

If you want to kill a process that is currently running, use the `kill` command.

```
kill -9 <PID(s)>
kill -9 2916
```

This command will kill your shell (see above). The `-9` flag should always be used to ensure that the process specified is killed (see man pages). Note that you can only kill processes which you own!

There is another command that may very useful when checking processes: `top`. This command lists the most active processes running on the host computer, along with their process ids, states, time running, CPU usage and process owner.

```
top
```

UNIX Remote Use:

We already saw some remote connection commands such as `telnet` and `ssh`. Another type of remote connection is the file transfer protocol (FTP). `Ftp` allows us to transfer files (binary or ascii) across file systems of different hosts.

```
ftp <hostname>
<username>
```

```
<password>

bin

prompt

lcd <desiredlocaldirectory>

cd <desirecremotedirectory>

mget *

bye
```

Ftp has a command line of its own, where ftp-commands are executed. Above are a sample to these. Here is a list of the commands we can use and what are they used for:

put <filename>	copies file from local computer to remote
mput <filenames>	same as put but allows several files and wildcards (prompt)
get <filename>	copies filename from remote computer to local
mget <filenames>	same as get but allows several files and wildcards (prompt)
cd <directory>	changes the working directory on remote computer
lcd <directory>	changes the working directory on local computer
bin	tells ftp to transfer files without translation
ascii	tells ftp to transfer files in plain text, translate if necessary
prompt	when off, allows the use of wildcards
bye	logout remote machine and exit ftp

With the advent of windows and window-like environments, many applications and programs are made to exploit the advantages of window environments (not many people like command line). Red Hat Linux now has a GUI utility called gftp, so you don't have to use the above commands. It's still faster to do if you know them.

If you are sitting in the console of a machine, all X display (UNIX's windows) will be placed in the console screen (0.0). However, if you want to run an X application remotely (such as matlab) and have the remote host place the windows in YOUR computer's screen (yes this can be done) we have to do the following. First, we must allow our host machine to accept X displays from other machines. Second, log into the remote computer. Third, instruct the remote computer to send the X display to the local computer.

Local machine,

```
xhost + <remotehostname>

telnet <remotehostname>
```

Remote host,

```
setenv $DISPLAY <localhostname>:0.0
```

This series of commands will allow us to get/see the display from remote computers.

As a note aside, if someone is logged in a machine and you wish to log into the same machine on console (not remotely) because your machine is not working properly, do this,

```
su <username>      or      su - <username>

ps -uga           or      top

kill -9 <PID(s)>

exit
```

The “-“ in the su command lets your shell know if your resource files should be loaded.

Another remote host check command is ping (should be in /usr/sbin).

```
ping <hostname>
```

No output from ping indicates that the remote host is not doing well.

UNIX Services:

Electronic Mail:

There are a variety of programs which have been written for electronic mail in UNIX (and other operating systems), for example, pine, netscape, ...

There is an e-mail utility found in almost all UNIX systems: mail

To send mail,

```
mail <address1> <address2> à
<type subject if prompted, may have to use ûs flag>
<type message>
<when done type . and <return> or ctrl-D>
```

Addresses are denoted by <username>@<email_hostname>

To navigate around the line you may need to use ctrl-p, ctrl-n, , ctrl-u, ...

Note that if there is a text file you would like to send to someone we can use:

```
mail <username@host> < <textfile>
```

The text last entries in the text files should be . <return> and <return> for this to work in one step, otherwise you will have to enter them to finish the execution of the mail command.

To read your mail,

mail

A message header is initially printed where messages are marked by new ("N"), unread ("U") or ("D"). Unread messages are those which were not read the previous time mail was checked and deleted messages are those marked for deletion after you quit.

The ">" denotes the current message. If you press <return> the current message will be displayed and the marker will move to the next message, so another <return> would show the next message.

Pressing "d" and <return> would mark the current message (">") to be deleted when quitting mail.

Messages that are not deleted will be saved in a file called "mbox" in your home directory.

To quit mail without changes use exit

To see the commands that can be used in mail type "h" or "?"

Here are some,

<#>	display message #
p	display present message
n	display next message
d	delete current message
r	reply to sender
h	see message headers
q	quit
exit	exit

The use of other mail programs is encouraged!

from

If you are in the host computer that serves as mail server, you can type prompt to see the senders of new messages

Text Editors:

Two prominent editors: vi and emacs

Nice feature about these editors is that they do not require a window manager for use, so they can be used from a telnet window from home!

Vi:

vi <filename>

If <filename> exists the contents of the file are going to be shown on the screen, if not when the session ends (with a successful save) a new file will be created with the contents entered and named <filename>.

vi is a bare bones text editor. It can be navigated with your keyboard arrows, however if these are not mapped correctly use g,h,j,k to move left, up, down, right. To navigate vi

must be in the command-mode, which is established by pressing <esc>. vi also has a command line of its own (ex commands used here) which is accessed using <esc>.

Commands to be used in command mode:

Arrow keys or ghjk	navigate
i	insert
a	append
dd	delete line
x	delete character
l	go to beginning of line
\$	go to end of line
5x	delete 5 characters
/<text>	forward find <text>
?<text>	backwards find <text>
n	find nextA
A	append at end of line

Commands in command line:

w	write file
wq	write file then quit
q!	quit without saving changes
w <filename>	write file as <filename>

Printing:

Files can be directed to the printer for hardcopy instead of the screen using the printing commands

Most systems use commands lp or lpr for printing

lp <filename> or lpr <filename>

request (<printername>-<id#>) sent

Flags to be used with lp or lpr commands

-n#	# copies
-d<printername>	redirect output destination to <printername>
-m	send email to user when print complete
-onb	no banner or title page for printout

What if the file is not printing? We can check the print jobs the printer is printing

lpstat -o or lpq

If your print job is not in the list it means it was either printed already or it never made it successfully to the printer

What if the printer is down and you want to cancel your 200 page document?

cancel <printername>-<id#> or lprm <id#>

Canceling print jobs is important because if the document you print is not understandable by the printer there is a chance the printer will attempt to print it and use up MANY pages. If a printer has gone down and the jobs are not cancelled, when it comes back up it will print them!