

Introduction to Unix

CS125/CS225
Fall 2004

This is a tutorial on some Unix system commands.

1 Getting Started on Unix

Before you can do anything else, you need to log in to Unix.

1.1 The computer laboratories for this course

Our labs are the **Engineering Workstation Labs** (EWS) run by the College of Engineering. Specifically, we will be using the EWS Sun machines. There is a wealth of information about the EWS labs available at the EWS web site, <http://www.ews.uiuc.edu>. You should all have accounts on the EWS network.

1.2 Logging into UNIX

You should all have accounts on the EWS network already; therefore, logging in should simply be a matter of obtaining your password and logging in at some machine. The EWS network uses your campus Active Directory password; the site to change this password is located at <http://accounts.ad.uiuc.edu/>. The password change site will ask you for your NetID and NetID Password for verification of your identity; once that is accomplished you will be presented with a screen where you can type in your new Active Directory password twice.

The assorted EWS labs are listed at <http://www.ews.uiuc.edu/labs/>. In this course, we will be using the UNIX (Solaris 8) machines, so please scroll down the page slightly, to where the listing of UNIX (Solaris 8) labs begin. If you go to one of those labs, and log in to one of those Solaris 8 machines, you should be able to complete the rest of this tutorial.

2 Creating a File

Now that your account is set up, use the **Emacs** editor to create the file ‘myname’. Below, and for the rest of the handout, the % indicates the prompt in your command-line window, at which you can type commands. So, if we tell you to type

```
% foo
```

 (1)

you are not supposed to type the % – that just represents the prompt at which you type a command, and you would type `foo` and hit return at that prompt. On some systems, your prompt might be something other than a %, such as perhaps a `>`, in which case meaning your command line might look like this after typing `foo`:

```
> foo
```

So, to begin the **Emacs** editor, type

```
% emacs myname
```

 (2)

You can instead type

```
% xemacs namname
```

 (3)

to start up the `XEmacs` editor the TAs might have demonstrated to you; `XEmacs` is just `Emacs` with more of a graphical wrapper around it. The instructions in the rest of this tutorial should work the same either way. (If you are already familiar with another text editor, such as `vi`, you may use it instead. This tutorial assumes you do not know how to use any text editor, and so attempts to teach you the basics of one – specifically, it attempts to teach you the basics of `Emacs`.)

A full description of `Emacs` is best left for another handout, but all you need to know for now is that you can insert text by just typing it into the window! Also, you can use the following keys to move around in the window, save your file, and exit `Emacs`. `C-x` means hold down the control key and press `x`. Two keypresses listed one after the other, such as `C-x c-s`, means you type them one after the other – i.e. first you hold down the control key while pressing `x`, and then you hold down the control key while pressing `c`.

<code>C-f</code>	forward character	<code>C-b</code>	backward character
<code>C-n</code>	next line	<code>C-p</code>	previous line
<code>C-a</code>	beginning of line	<code>C-e</code>	end of line
<code>C-k</code>	delete line		
<code>C-x C-s</code>	save file	<code>C-x C-c</code>	leave Emacs

Type in some text now (perhaps your name and someone else’s name on two lines, though whatever else you like instead is fine too). When you are finished, type `C-x C-s` to save your file and exit `Emacs`.

3 Simple Unix Commands

3.1 Entering Commands

When entering commands, remember that Unix is *case-sensitive*. In most cases, you must enter the name of a command using all lower-case letters. User names and passwords are also case-sensitive. If you make a mistake when typing something in, you can generally correct it by using the backspace key to delete the errant characters. To delete an entire line, use Control-U.

3.2 Listing and Copying Files

Type

```
% ls (4)
```

to list the files in your account. You should see `myname` listed. A second way to create files is by copying; type

```
% cp myname other (5)
```

```
% ls (6)
```

to create a copy of `myname` in `other` and view the results. You can also look at a “long” listing which gives more information about the two files; type

```
% ls -l (7)
```

You should see something like

```
total 32
-rw----- 1 zych    facl      25 Aug 31 07:49 myname
-rw----- 1 zych    facl      25 Aug 31 07:49 other
```

The first field gives information about who can read and modify the files; this will be discussed in detail later. You can also ignore the second field (the “1” above) for now. The third field is your username. Looking more toward the right of the listing above, you see `Aug 31 07:49`; that is the date and time the file was last modified. The number to the left of it (25 in the example above) relates to the size of the individual file – since you created `other` by making it a copy of `myname`, both the files are the same size right now. Finally, the last field gives the name of the file. (The line “total 32” at the top, you can ignore for now.)

You may also have some *dotfiles* (files that have names beginning with “.”) in your account. Type

```
% ls -a (8)
```

to see the dotfiles in your current directory. The names of files beginning with “.” are not printed by `ls` unless you use the `-a` flag. Dotfiles are often used for configuration files and other such things that you are not likely to want to use all the time. This way, you can use `ls` to view your work files, and aren’t forced to have to wade through a listing of configuration files as well.

3.3 Viewing Files

One way to view a file is to use Emacs. But if you just want to take a quick peek at the file, another way to view files is to use the program `more`. Try the command:

```
% more other (9)
```

and then try the command:

```
% more myname (10)
```

You should see the same thing both times, since one file is a copy of the other.

If the file is longer than a page, the `more` command displays a file on the screen one “page” (that is, one screenful) at a time. When it has printed the first page, it pauses and prints `--More--` at the bottom of the screen. Press **space** to advance a page and **return** to advance by one line. In some cases, `more` will also print the percentage of the file that you’ve viewed so far. Try it out:

```
% more /usr/local/ews/system.cshrc (11)
```

3.4 Renaming Files

In addition to copying files, you can also rename them using the `mv` command (“mv” is short for “move”):

```
% mv other junk (12)
```

```
% ls (13)
```

3.5 Deleting Files

Finally, you can delete files by using the `rm` command (short for “remove”), though the default on some Unix systems is for the `rm` command to prompt you to type “yes” or “no” as a double-check that you actually do want to delete the file. Type

```
% rm junk (14)
```

```
% ls (15)
```

to delete the file you just created above and see the result. You should now just see the file `myname`. You can remove multiple files at the same time:

```
% cp myname junk (16)
```

```
% rm junk myname (17)
```

`ls` should now show there are no (non-dot) files in your account (assuming you haven’t created other files!).

3.6 Getting Help

One of the features of Unix is that most commands have online documentation. The `man` command is used to read them. Type

```
% man more (18)
```

Note that `man` uses the `more` command to page through the documentation. Try also

```
% man man (19)
```

The `apropos` command is used to find commands by keyword; type

```
% man apropos (20)
```

Part of the reason we supply a handout like this to you, is that manual pages such as the above can be rather cryptic to new Unix users. However, once you are more comfortable with Unix, manual pages like the above can be used to learn more about the system. A computer is a tool, and just as with any other tool, if you want to learn to use it well, you need to be willing to spend some time with it. We are not saying you should be reading lots of manual pages right now, but eventually, you want to learn a bit more about the system than this tutorial covers.

4 More On Unix: Directories

A *directory*, contains some set of files. (A folder in Mac OS X or Windows, is merely a graphical representation of a directory.) Each user has his or her own directory; this way different users can have files with the same name without creating problems. Another advantage of separating accounts into separate directories is that directories can be protected so that only the owner of a directory can view its contents.¹

In addition to regular files, directories can contain other directories. The result is that the Unix file system is organized like a genealogical tree. The “top” of the tree, known as the *root*, essentially contains all the files that are on the machine in various subdirectories, which can differ from network to network. On EWS, one of these subdirectories is “`home1`” where all the user accounts exist whose netIDs start with the letters `a-k`. The directory `home2` contains all the user accounts whose netIDs start with the letters `l-z`. If you type

```
% ls / (21)
```

you can see the complete listing of everything in the top-level, or root, directory.

As we said before, each user has their own directory in the system. This original directory is known as your *home directory*, to distinguish it from other directories you might create within the home directory. (The analogy to a graphical interface like Windows or Mac OS X is a top-level folder, inside which you can create other folders.) For example, the full name for the home directory of the user `zych` is

```
/home2/z/zy/zyc/zych
```

Note that the various directory names are separated by slashes (`/`). The complete name is known as a *path name* since it specifies the path from the root to a particular directory. Two other useful paths name are `/homesta/cs125` and `/homesta/cs225`, which is where the class-related files for CS125 and CS225, respectively, are located.

You can give the name of a directory to `ls` to see what’s in it. Try

```
% ls /home/class/cs125 (22)
```

¹But note that system administrators can view any file. However, good system administrators will not look at an individual user’s files except in extreme cases. In particular, your EWS account has been set up this way – while the system administrators can look at your files, no one else can unless you specifically set things up to provide access to others.

It's pretty difficult to remember something like `/home2/z/zy/zyc/zych`, even if it happens to be the name of your home directory. Fortunately, there are two things that aid you in this regard. Firstly, when you log in, your current directory is automatically set to your home directory. (We talk in a moment about setting your current directory yourself.) That is, when you log in, you are automatically in your own account's top-level directory, rather than being placed into the root directory of the system. Secondly, there's an easy-to-remember abbreviation for your home directory: `~`. Thus `ls ~` lists all the files in your home directory except for the hidden files, and `ls -a ~` will list all the files in your home directory including the hidden files.

In general, the home directory of user `xxx` is given by `~xxx`; type

```
% ls ~X (23)
```

where `X` is your login name. You can also try

```
% ls ~cs125 (24)
```

to list the files in the home directory of the `cs125` account, and

```
% ls ~cs225 (25)
```

to do the same for the `cs225` account.

The `cd` command allows you to change what directory you are in. Type

```
% cd /homesta/cs225 (26)
```

```
% ls -l (27)
```

This directory contains several subdirectories; the first field of the long listing shows which are directories by starting with a `d`. Note that `src` is a subdirectory; type

```
% cd src (28)
```

In this case, `src` is what's known as a *relative* path name since it is relative to your current location. In contrast, a name like `/homesta/cs225/src/` is known as an *absolute* path name. Type

```
% ls (29)
```

to see the names of various directories that are within the directory `/homesta/cs225/src/`.

To see what directory you are in, type

```
% pwd (30)
```

which stands for "present working directory". This command should output `/homesta/cs225/src`, since that is the directory you are in right now. (The analogy to a graphical interface is that `pwd` tells you what folder you are in.)

The absolute paths of some other example directories you could look at are:

/bin The directory containing a number of Unix utilities, including `ls`, `more`, `cp`, and `rm`.

/homesta The course directories for each course whose students have course accounts on the EWS machines.

/lib The directory containing assorted software libraries

We'd like to encourage you to use `cd` and `ls` to browse around the system. You can learn a lot about Unix that way. However, be cautious about what commands you run; some commands might have unexpected effects, so read the documentation first, before running an unfamiliar command. (The commands we have talked about here, you can consider familiar.) In addition, you will probably see dot files in your home directory, with names like `.cshrc` and `.login` and such. Do NOT delete these. As we mentioned earlier, the dot files are generally used for system configuration; if you delete the wrong files, you can seriously mess up your user account. So, don't delete a dot file unless you know for certain what it is and know for certain that it is safe to delete that file.

To return to your home directory, type `cd` with no arguments:

```
% cd (31)
```

Note that you could also type `cd ~` to return to your home directory.

In Unix, a file can have multiple names in different directories. Such multiple names are known as “links.” As an example, type

```
% ls -la (32)
```

Note the very first two names that are listed: “.” and “..”, and note that both of these are directories (by looking for the `d` at the beginning of the line). “.” and “..” are examples of links. “.” is another name for your current directory; thus typing

```
% more ./cshrc (33)
```

is equivalent to “`more .cshrc`” (assuming you are in your home directory) in that it displays out the contents of your `.cshrc` file. “..” is another name for the directory *above* your current directory. Type

```
% cd .. (34)
```

```
% pwd (35)
```

to move up one directory towards the root and see where you are. (“Moving up one directory” is the equivalent of, in a graphical interface, moving into the folder that *contains* whatever folder you are in right now.)

4.1 Creating Directories

It is usually convenient to arrange your own files so that all the files related to one thing are in a directory separate from other projects. For example, if you want to do a project `mp1`, you might want to keep all the text in a directory called `mp1`. The command `mkdir` makes a new directory; type

```
% mkdir mp1 (36)
```

```
% ls (37)
```

(assuming you don’t already have a `mp1` subdirectory; if you do, use a different name). Then type

```
% cd mp1 (38)
```

```
% pwd (39)
```

and use Emacs to create a file (which can contain anything you want).

Now type

```
% cd .. (40)
```

```
% ls -l (41)
```

to list the contents of your home directory (which will probably just be the directory `mp1` unless you’ve added other things as well) and

```
% ls -l mp1 (42)
```

to list the contents of the directory `mp1` that is in your present working directory (and since your present working directory is your home directory, what you are doing is listing the contents of the `mp1` directory that is inside your home directory).

4.2 Unix Filenames

A Unix filename can be almost any string of characters. Most of the time you will use letters, numbers, and dashes. In Unix, case matters, so the file name `MyFile` is different from the file name `myfile`. There is a limit to how long a file name can be, but it's such a large limit that you're very unlikely to run into it (it's around 1000 characters).

In some ways, it is useful to think of a pathname as the full name of a file and the various relative names (including just the name of the file with no directories) as abbreviations. So, for the account `cs125`, whose home directory is `/homesta/cs125/`, since that directory contains the file `noteToStudents`, you can think of `/homesta/cs125/noteToStudents` as the full name of the file, and `noteToStudents` as an abbreviated file name, which leaves off the part of the full name that indicates how to get to the directory where the file is located.

All Unix commands will take a pathname (either absolute or relative) in the place of a simple file name. Hence most of the online documentation (the “man pages”) uses “filename” to mean any pathname. Furthermore, a directory is actually just a special type of file, and so many commands work on both directories and files:

```
% cd (43)
```

```
% ls -l .cshrc (44)
```

```
% ls -l mp1 (45)
```

But some commands won't allow you to use directory names; try

```
% more mp1 (46)
```

and it won't work. The `ls` commands worked because it's possible to list a file (in which case only the name of that file is listed) or to list a directory (in which case the contents of that directory are listed), but even though you can read a file (using `more`), you cannot “read a directory” since a directory is nothing but a container for other items and so there's nothing specific to list with the `more` command.

4.3 Simple Unix Commands and Directories

Several commands have special behavior when you give the name of a directory instead of a file. This section looks at four of these: `cp`, `mv`, `ls`, and `rm`.

If the second argument to `cp` is the name of a directory, then the file is copied to that directory with the same name. For instance, try

```
% cp .login mp1 (47)
```

```
% ls -a mp1 (48)
```

You should see the file `.login` listed inside `mp1`.

This is especially helpful if you want to copy a file to your current directory, using “.” as the destination, since this way you can avoid retyping the full name. Try it:

```
% cd mp1 (49)
```

```
% cp ../cshrc . (50)
```

```
% ls -a (51)
```

```
% cd (52)
```

However, if the *source* file name to `cp` (that is, the first name listed) is a directory, then `cp` will not let you do the copy:

```
% cp mp1 new-mp1 (53)
```

If you really do want to copy an entire directory, use the `-r` flag (for a *recursive* copy):

```
% cp -r mp1 new-mp1 (54)
```

```
% ls (55)
```

`mv` is similar to `cp`. If a directory is given as the second argument, the file is actually moved:

```
% cd (56)
```

```
% cp .cshrc junk (57)
```

```
% ls (58)
```

```
% mv junk mp1 (59)
```

```
% ls (60)
```

```
% ls mp1 (61)
```

But if a directory is given as the first argument (as well as the second), it is moved:

```
% mv new-mp1 mp1 (62)
```

```
% ls (63)
```

```
% ls mp1 (64)
```

We've already seen that if `ls` is given a directory name instead of a file, the contents of that directory are listed. But when you want to see the files in the subdirectories of that directory, it can get tedious to type in all the directory names. Instead, you can use the `-R` flag for `ls`:

```
% cd (65)
```

```
% ls -R mp1 (66)
```

Generally, `rm` cannot be used to remove a directory:

```
% cd mp1 (67)
```

```
% rm new-mp1 (68)
```

Instead, you must use `rmdir`. However, you cannot remove a directory that contains a file in it:

```
% rmdir new-mp1 (69)
```

You must first delete all the files that are in the directory:

```
% cd new-mp1 (70)
```

```
% ls -a (71)
```

and then use `rm` to remove all the listed files (except `.` and `..`). Then type

```
% cd .. (72)
```

```
% rmdir new-mp1 (73)
```

A faster way to do this is to use the recursive flag for `rm`. But note that this is a dangerous command since you can delete a lot of files very quickly. Once a file is deleted in Unix, it cannot be recovered. Assuming you've been using the name `mp1` in all the above, type

```
% cd (74)
```

```
% rm -r mp1 (75)
```

to clean things up.

4.4 Directory Management

If you aren't used to using directories, it may be tempting to just ignore them and do everything in your home directory. *Don't*. It's almost impossible to keep track of what you're doing when you have 30 files representing different stages of different projects in the same directory. *Always* create a separate directory for each project (lab work, MPs, etc.). Not only will it be less confusing, but you'll also be less likely to destroy your work by accident.

Likewise, it's usually a bad idea to keep lots of versions of programming solutions laying around in the same directory. If you want to keep a recent backup version while making a lot of changes (which is not a bad idea), make a subdirectory `bak` and put a copy of your file there. That way you can always be sure what file contains the latest version of your solution, since you've put the backup copies elsewhere, and it is easy to delete old versions (old backups) once you have a working solution.

5 Logging Out

Be sure to log off before leaving the lab. Leaving the terminal while you are logged on is inviting trouble! Someone might delete your files or print up a programming solution. To log off, see the “[Quitting X-Windows and Logging Off](#)” item in Section 2.

If you have any questions, you can ask us or post to the newsgroup and we can help you out.

6 Credits

This handout was written by Ken Kruska in the late 90s, and modified more recently by Jason Zych. Ken might have based his handout on someone else's work, but if so, that information has been lost to the mists of time.