An Introduction To UNIX*

Pinghua Young
and
Grace Katagiri

October 30, 1995

1 Getting Started with UNIX

1.1 Logging In

What Is Logging In? Logging in is the process by which you identify yourself to a UNIX system. Before you begin a session at the console, you must use your UNIX account to log in to a workstation (emily3,...,emily14). Logging in simply is the process through which you tell the system your username and your password.

Why Doesn't My Password Appear on the Screen? Notice that your password does not appear on the screen as you type it. This is just added security provided by the UNIX system, so that curious eyes behind you cannot read your password. Never share your password with anyone, and don't write it down either. (In the EML, if we find that account information has been shared, the account is immediately and permanently revoked.)

What about Typing Mistakes? If you make a mistake, you must type both your login name and password again. You are given about three to five more chances to type them incorrectly (depending on the system), after which you will be disconnected. Once you successfully log in, the system displays the command-line prompt and waits for your commands. If you log in from the console (meaning you are sitting in front of an emily in room 616 Evans), you will default to the SunOS Open Windows environment. You can also use the generic X Windows system if you wish. A windows system is a collection of software that makes it easy for you to take full advantage of the graphics and windowing capabilities of a workstation. The following is a sample login session. (This is a dial-up from home example, so it doesn't go into a graphical windows system automatically.)

```
login: pinghua
Password:
Last login: Mon Jan 11 22:45:26 from emily8.Berkeley.
SunOS Release 4.1.1 (ECON_SSx) #5: Wed Aug 5 12:07:41 PDT 1992
You have new mail.
TERM = (vt100)
Erase set to Backspace

>> STUDENT ACCOUNTS MUST BE RENEWED FOR SPRING SEMESTER PRIOR TO
>> 1/19/93.  See "news" for details.
```

*Much of the stuff included here is taken from various materials gathered over the years. If you want to know more, read the book *C Shell Field Guide* by Anderson and Anderson.

```
>> The workstations in 616 are reserved for tutorials 1/12-14/93
>> (Tu-Fr) from 10:00 a.m. - noon.

>> The air conditioning work in the EML is finished.  All the machines
>> are running as usual.

>> The EML is reserved for graduate class use M-F 12:00 n. - 5:00 p.m.


Type "help" for help and "news" for news (updated Fri  8 Jan 1993).

Load: 10:46pm  up 4 days,  6:26,  0 user,  load average: 0.14, 0.02, 0.00

Date: 01/11/93  Time: 22:46:24
Last logout: Mon Jan 11 20:24:52 PST 1993
501 emily10>
```

Notice the prompt is **501 emily10**>, which is likely to be different from your prompt. Your default prompt will have a percent (%) sign, which is the prompt for the C shell. (Most UNIX systems default to the C shell.) Whenever you see the prompt, you know the system is waiting for further instructions from you.

1.2 Changing Your Password

When? Every once in a while, you definitely want to change your password to preserve the security of your account. If you do not change your password for a long time, the system might also refuse your login request. If you are using a class account, then the first time you login you should change your assigned password immediately. You should also use the **chfn** command to put your real life name and phone number into the system.

Can I Change My Username? No. Once your choose your username when opening your account, it cannot be changed easily. Your username is like your personal name, and it will follow you throughout your career at Berkeley.

Changing Your Password—Your password must *not* be any word found in the English dictionary, your username, or any permutation of your name or initials. To change your password, use the command **passwd**. Remember, your password does not appear on the screen when you type it.

```
520 emily8> passwd
Changing NIS password for pinghua on econ.Berkeley.EDU.
Old password:
New password:
Retype new password:
NIS entry changed on econ.Berkeley.EDU
521 emily8>
```

1.3 Logging Out

What Is Logging Out? Logging out ends your session. You must finish each work session by logging out so that no one else can continue working on the system *in your account*. If a message on the screen indicates there are stopped jobs, be sure to 'kill' them before logging out. See Section 5.7 for more information on job control and related commands to kill a job or process.

If you are at the console of a Sun, move your mouse pointer to a blank (non-window) area of the screen, click with the rightmost button, and select "Exit". You will be prompted to confirm "Exit". This will log you out completely. If you are in a terminal emulation (character-based, shell, non-windows), just type "logout" or "exit" to end your session.

```
522 emily8>  logout
```

1.4 Getting Information

The UNIX Manual—On the EML system, the complete UNIX reference manual is online (the printed version takes up roughly five linear feet and costs close to $2,000). The manual contains descriptions of the UNIX commands online so that you can refer to it on your screen as needed. Unfortunately, you may have to dig around to find just what you're looking for. The **man** and **apropos** commands can help you. The online manual is organized by command; as a novice, you may find that the difficulty is knowing which command you need to read about. As shown in the following example, you can use the **-k** option for **man** to obtain a list of topics related to a particular keyword. Also, printed copies of part of the UNIX manual and of the SUN workstation manuals are located in 616 Evans for easy reference.

```
527 emily8> man man



MAN(1)                    USER COMMANDS                    MAN(1)



NAME
     man - display reference manual pages; find  reference  pages
     by keyword

SYNOPSIS
  .
  .
  .
529 emily8> man -k password
xlock (1)               - Locks the local X display until a password is entered.
  .IX xlock#(1) "" "\fLxlock(1)"
xlock (1)               - Locks the local X display until a password is entered.
 .sp
chkpw (8)               - check password file entries against reality
pwget, grget (UTIL)     - get password and group information
crypt, _crypt, setkey, encrypt (3)  - password and data encryption
getpass (3V)            - read a password
getpwaent, getpwanam, setpwaent, endpwaent, fgetpwaent (3)  - get password adjun
ct file entry
getpwent, getpwuid, getpwnam, setpwent, endpwent, setpwfile, fgetpwent (3V) - ge
t password file entry
passwd (5)              - password file
passwd, chfn, chsh (1)  - change local or NIS password information
putpwent (3)            - write password file entry
pwck (8V)               - check password database entries
pwdauth, grpauth (3)    - password authentication routines
```

```
pwdauthd (8C)           - server for authenticating passwords
rfpasswd (8)            - change RFS host password
vipw (8)                - edit the password file
yppasswd (1)            - change your network password in the NIS database
yppasswd (3R)           - update user password in NIS
yppasswd (5)            - NIS password file
yppasswdd, rpc.yppasswdd (8C)   - server for modifying NIS password file



531 emily8> man passwd



PASSWD(1)                USER COMMANDS                   PASSWD(1)



NAME
     passwd, chfn, chsh - change local or NIS  password  informa-
     tion

SYNOPSIS
    .
    .
    .
532 emily8>
```

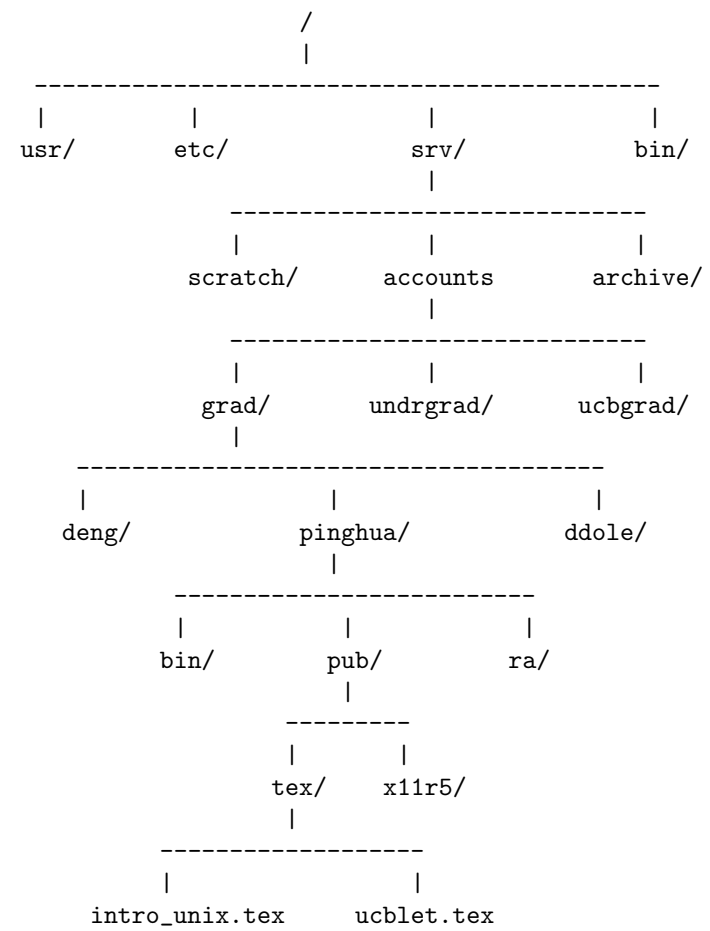
If you are using Open Windows or the X Window System, you can also use the **xman** command, which has a much nicer user interface.

2 Working With Files and Directories

2.1 The UNIX File System

What Are Files? You can think of computer files as the folders inside a file cabinet. Each file has a name. Files are quite important because they allow you to store information for use at a later time. After you finish your session, the system forgets what you've been doing, unless you save your work in a file "to disk.".

How Is a File System Organized? UNIX files are organized into groups called directories (they are also called folders). Directories are files, but instead of containing data or text as other files do, the directory contains other files. The directories are organized like an inverted tree. The root directory named / is at the top. Note that the root directory in a UNIX system is not equivalent to the PC concept of "root." For PC users, think of your "root" directory as being equivalent to the UNIX "user's home directory." The user's home directory usually starts well below the root and other system directories in a UNIX filesystem tree. Branching out from the root directory are the rest of the directories. Any directory can contain both directories and other files. Below is a diagram of a simplified filesystem. The filesystems on the EML system are more complex than the fictional one depicted below.



2.2 Directories and Paths

The Root Directory—The / at the top of the illustration is the root directory. Within this root directory are: **usr, etc, srv**, and **bin**. These files just happen to be directories, which contain their own files. These directories-within-a-directory are called subdirectories. Thus, **srv** is a subdirectory of /. Every file on the system is contained in the root directory or in one of its subdirectories.

Your Home Directory—When you first login to a UNIX system, you are in the directory that holds your personal files. This directory is called your home directory. In the example above, the home directory of the user named *pinghua* is **/srv/accounts/grad/pinghua**. If you are also a graduate student in the Economics Department, your home directory will be **/srv/accounts/grad/your_username**.

Your Current Working Directory—Whenever you use a UNIX system, you are always working in a specific directory (could be your home directory; could be a subdirectory; could be a public directory elsewhere on the system). That directory is called the current working directory, or simply, the *working* directory. For example, if you were in the **/usr** directory, your working directory would be **/usr**. If you were in **/srv/accounts/projects**, then your working directory would be **/srv/accounts/projects**. Most commands that operate on directories use the working directory as the default, unless you specify otherwise. Therefore, it's important to know what the working directory is; to find out, use the **pwd** command (you can think of **pwd** as **p**resent **w**orking **d**irectory):

```
539 emily8> pwd
/tmp_mnt/srv/accounts/grad/pinghua/pub/tex
540 emily8>
```

2.3 Finding Out What's in a Directory

The Directory Listing Command—The command for listing the files and subdirectories contained in a directory is **ls**. As shown in the following example, different information is displayed, depending on what arguments you use.

```
542 emily8> ls
ddole                   intro_unix.tex          short_loc_gid.tex.Z
intro_unix.aux          local_guide.tex.Z       ucblet.tex
intro_unix.dvi          multi_ucblet.tex.Z
intro_unix.log          multi_ucblet_body.tex.Z

543 emily8> ls /usr
5bin            custom          kvm             openwin         ucbinclude
5include        demo            lang            pub             ucblib
5lib            diag            lib             sccs            wp
adm             dict            local           share           xpg2bin
app             dist            lost+found      spool           xpg2include
bin             etc             lpp             src             xpg2lib
blss            export          man             stand
boot            games           mdec            sys
cchelp          hosts           new             tmp
crash           include         old             ucb

544 emily8> ls -l
total 63
-rw-------  1 pinghua      6186 Aug 25 11:43 ddole
-rw-------  1 pinghua       225 Jan 11 17:16 intro_unix.aux
-rw-------  1 pinghua      2800 Jan 11 17:16 intro_unix.dvi
-rw-------  1 pinghua       943 Jan 11 17:16 intro_unix.log
-rw-------  1 pinghua     13745 Jan 11 23:57 intro_unix.tex
-rw-r--r--  1 pinghua     18401 Mar 12  1992 local_guide.tex.Z
-rw-r--r--  1 pinghua      1040 May 11  1992 multi_ucblet.tex.Z
-rw-------  1 pinghua       843 May  9  1992 multi_ucblet_body.tex.Z
-rw-r--r--  1 pinghua      7514 Feb 24  1992 short_loc_gid.tex.Z
-rw-r--r--  1 pinghua      7483 Oct 10 17:41 ucblet.tex

545 emily8> ls -F
ddole/                  intro_unix.tex          short_loc_gid.tex.Z
intro_unix.aux          local_guide.tex.Z       ucblet.tex
intro_unix.dvi          multi_ucblet.tex.Z
intro_unix.log          multi_ucblet_body.tex.Z
546 emily8>
```

Dot Files—Within your home directory are some files whose names begin with a dot (or period). Most users don't need to access dot files often. They don't show up on the list supplied when you type the command **ls** (unless you use the option **-a**). Because they aren't listed on the screen of your terminal, you can find other files more easily. Here is a list of some dot files:

.cshrc The system runs the commands in this file each time you start a shell. Usually a good place for alias commands. See Section 3.1 for more information.

.forward Contains address(es) where mail for this account should be forwarded, e.g., jbai@athena.mit.edu. This is useful if you want to receive your mail at a different location instead of at the EML system. Also used with the vacation mail program (see "help vacmail").

.login The system runs the commands in this file when you login. Usually you don't need to change anything here.

.logout The system runs the commands in this file when you logout.

.plan An optional file you create that contains personal or work-related information of an arbitrary nature; displayed when your username is *fingered*.

.project An optional file you create that is limited to a short, one line description of your project or course work; displayed when your username is *fingered*.

```
548 emily8> ls -a
.                       .login                  .vacation.dir
..                      .logout                 .vacation.msg
...                     .mailrc                 .vacation.pag
..forward               .mailtool-init          .wastebasket
..plan                  .newsrc                 .xinitrc.x
..rhosts                .newsrc.ucbvax          .xinitremoterc
.Data                   .oldnewsrc              .xloadimagerc
.Splus_history          .openwin-init           .xsession
.Xauthority             .openwin-init.BAK       .xsession-errors
.Xdefaults.x            .openwin-menu           News
.Xresources             .plan                   adm
.Xresources.bak         .plan.gb                bin
.article                .plan.hz                books
.aut                    .plan1                  calendar
.cetables               .plan2                  chinese
.cshrc                  .pnewsexpert            lib
.desksetdefaults        .pobox                  login
.fexp                   .project                pub
.forward                .rnlast                 sounds
.gopherrc               .rnsoft                 stat
.history                .sh_history             tex
.kermrc                 .signature              thesis
.l123set                .std.login.mpy          tmp
.lastlog                .texrc                  unix
.letter                 .twmrc
549 emily8>
```

2.4 Changing Directories

You use the **cd** command to change to another directory. Just "cd" by itself always returns you to your home directory. If you use "cd" with one dot (.), it means change to your current working directory, and two dots (..) means the parent directory that contains your current working directory. So "cd ." is relatively meaningless, but "cd .." will move you up the directory tree, and "cd" alone will always take you home.

```
556 emily8> pwd
/tmp_mnt/srv/accounts/grad/pinghua/pub
557 emily8> ls -F
tex/    x11r5/
558 emily8> cd tex
559 emily8> pwd
/tmp_mnt/srv/accounts/grad/pinghua/pub/tex
560 emily8> cd ..
561 emily8> pwd
/tmp_mnt/srv/accounts/grad/pinghua/pub
562 emily8>
```

2.5 Creating Directories

For organizational purposes, you may want to create a new subdirectory to hold a group of related files, as shown in the following sample session. The command for creating a directory is **mkdir**. After the new directory is created, you can either create files in that directory or move or copy existing files into it.

```
566 emily8> mkdir thesis
567 emily8> cd thesis
568 emily8> ls
569 emily8> cd ..
570 emily8>
```

2.6 Removing Directories

You can remove a directory by using the **rmdir** command. Before you do so, make sure the directory is empty, i.e., it contains nothing. See Section 2.7 for information about removing files.

```
572 emily8> ls thesis
573 emily8> rmdir thesis
574 emily8> ls thesis
thesis not found
575 emily8>
```

Absolute Pathnames—To identify a certain file in the file system, you can specify an absolute pathname by telling UNIX where the file is in relation to the root directory. For example, **/etc/test** is a file called **test** located within the **/etc** directory under the root directory. The first / signifies the root directory. Each subsequent directory name is separated by an additional /.

For more advanced users: Note that on the EML system, **all** absolute pathnames on the fileserver start with "/srv/...". If an absolute pathname starts simply with "/", then the pathname refers to a filesystem on the local workstation (for example, "/tmp" is swap space on the workstation's hard disk, and /data1/csrp is a filesystem that is NFS mounted directly on the workstation, not the fileserver.

Another arcane piece of information: Although "pwd" returns pathnames beginning with "tmp_mnt" on the EML system, you can ignore it in giving absolute pathnames. "tmp_mnt" is a vestige of the automounting facility in SunOS, and has no real physical existence.

Even more arcane: "/srv" is only a pointer to "/econ", which is the name of our primary fileserver. "/" is therefore the root directory of the fileserver "econ". Since we have more than one server, however, we use

"/srv" in pathnames because it always points to the correct server, even though files may move back and forth. For example, "/srv/scratch" is equivalent to "/econ/scratch" when logged into the "econ" fileserver. On the "emlab" server, "/srv/scratch" is equivalent to "/emlab/scratch".

'Nuff said.

Relative Pathnames—Besides identifying a file by its absolute pathname, you can use a *relative* pathname. That is, you can tell UNIX where the file is in relation to the working directory. Suppose we are in Pinghua's home directory that contains a subdirectory called **pub**. **pub** contains a file called **test**. Instead of using absolute pathname like **/srv/accounts/grad/pinghua/pub/test**, we can simply use **pub/test** to refer to it.

```
581 emily8> pwd
/tmp_mnt/srv/accounts/grad/pinghua
582 emily8> ls pub
test    tex     x11r5
583 emily8> cat pub/test
This is a test file.
584 emily8>
```

Like one dot (.) and two dots (..), there is also a shorthand for referring to home directories. ~ denotes your own home directory, while **~username** denotes **username**'s home directory. For example, **~pinghua** refers to Pinghua's home directory, so to go to his home directory from your home directory, you can simply type **cd ~pinghua**.

2.7 Handling Files

Copying Files—The command to copy files is **cp**. Its first argument is the source file, that is, the file you are copying. Its second argument is either the name you want for the new, copied file or the destination directory. If a directory is given as the second argument, the file is copied into that directory with the same file name. You can also use wildcards to copy a group of files. See the end of this sub-section for more information about wildcard characters.

Warning: When you use the **cp** command, the system doesn't check to see if a file already has the name you chose for the destination file. If that file name already exists, the system removes the existing file before it copies the file you specified. If you want the system to ask you for confirmation, use **cp -i** instead of simply **cp**.

```
601 emily8> ls
ddole                   intro_unix.tex          short_loc_gid.tex.Z
intro_unix.aux          local_guide.tex.Z       ucblet.tex
intro_unix.dvi          multi_ucblet.tex.Z
intro_unix.log          multi_ucblet_body.tex.Z

602 emily8> cp intro_unix.tex intro_unix_bak.tex
603 emily8> ls
ddole                   intro_unix.tex          multi_ucblet_body.tex.Z
intro_unix.aux          intro_unix_bak.tex      short_loc_gid.tex.Z
intro_unix.dvi          local_guide.tex.Z       ucblet.tex
intro_unix.log          multi_ucblet.tex.Z

604 emily8> cp intro_unix ..
cp: intro_unix: No such file or directory
```

```
605 emily8> cp intro_unix.tex ..
606 emily8> ls ..
intro_unix.tex  test            tex             x11r5
607 emily8>
```

Moving Files or Changing Their Names—The command for both moving a file from one directory to another and renaming a file is **mv**. Its first argument is the source file, and its second argument can be either the destination directory or the new name of the file.

```
608 emily8> ls
ddole                   intro_unix.tex          multi_ucblet_body.tex.Z
intro_unix.aux          intro_unix_bak.tex      short_loc_gid.tex.Z
intro_unix.dvi          local_guide.tex.Z       ucblet.tex
intro_unix.log          multi_ucblet.tex.Z

609 emily8> mv intro_unix_bak.tex /tmp
610 emily8> ls intro_unix_bak.tex
intro_unix_bak.tex not found

611 emily8> ls /tmp/intro_unix_bak
/tmp/intro_unix_bak not found

612 emily8> ls /tmp/intro_unix_bak.tex
/tmp/intro_unix_bak.tex

613 emily8> mv intro_unix.log garbage
614 emily8> ls intro_unix.log garbage
intro_unix.log not found
garbage
615 emily8>
```

Referring to Groups of Files—There are some so-called wildcard characters of which you should be aware. They provide you with a powerful way of referring to a group of files easily. One is the character *****, which means any string of characters. Another one is **?**, which refers to any single character. For example:

```
616 emily8> ls
ddole                   intro_unix.tex          short_loc_gid.tex.Z
garbage                 local_guide.tex.Z       ucblet.tex
intro_unix.aux          multi_ucblet.tex.Z
intro_unix.dvi          multi_ucblet_body.tex.Z

617 emily8> ls *.tex
ddole           intro_unix.tex  ucblet.tex

618 emily8> ls *.Z
local_guide.tex.Z       multi_ucblet_body.tex.Z
multi_ucblet.tex.Z      short_loc_gid.tex.Z

619 emily8> ls intro_unix.???
intro_unix.aux  intro_unix.dvi  intro_unix.tex
```

```
620 emily8> ls intro_unix.*
intro_unix.aux  intro_unix.dvi  intro_unix.tex

621 emily8> ls intro_unix.?
ls: No match.
622 emily8>
```

Creating and Removing Files—There are a lot of ways to create files. An obvious way is to use an editor such as **vi, emacs**, or **jove**, etc. You can also use UNIX's powerful output redirection to create a file. See Section 4 for more information about this and other powerful tips and tricks like pipes.

To remove a file, use the **rm** command. Be careful with this command. Once the file is removed, it's gone forever. You can't use Norton Utilities to get it back. Just to be on the safer side, use **rm -i** instead of plain **rm**. Or better yet, put **alias rm rm -i** into your **.cshrc** file.

2.8 Browsing through Files

Looking at Files One Screenful at a Time—You can use the **more, less**, or **page** commands to display a screenful of the file at a time. To display the next screenful, press the space bar. Type **man more**, **man less** for more information.

The UNIX command **cat** also allows you to view files on your screen, but it won't show one screenful at a time. However, you can use it for other purposes. For example, if you want to append **file2** to **file1**, simply type **cat file2 >> file1**. For more information, type **man cat**.

Looking at the Beginning or End of a File—You use the commands **head** or **tail** to display the beginning or the end of a file. The default is the first 10 or the last 10 lines, but you can specify how many lines you want, as shown in the following example.

```
649 emily8> head ~/.cshrc
#       source the standard .cshrc file
source  /usr/local/skel/std.cshrc

#       personal customization
set path = ($HOME/bin ~choices/bin $HOME/unix/mac /moby/pbmplus/bin /tmp_mnt/mob
y/b/tiff/sun4_sunos4.1.1/bin $path /usr/demo/SOUND /usr/local/x11r5/games /usr/e
tc)
source $HOME/.../.cshrc
umask 077

650 emily8> tail ~/.cshrc
#       source the standard .cshrc file
source  /usr/local/skel/std.cshrc

#       personal customization
set path = ($HOME/bin ~choices/bin $HOME/unix/mac /moby/pbmplus/bin /tmp_mnt/mob
y/b/tiff/sun4_sunos4.1.1/bin $path /usr/demo/SOUND /usr/local/x11r5/games /usr/e
tc)
source $HOME/.../.cshrc
umask 077

651 emily8> tail -5 ~/.cshrc
```

```
#       personal customization
set path = ($HOME/bin ~choices/bin $HOME/unix/mac /moby/pbmplus/bin /tmp_mnt/mob
y/b/tiff/sun4_sunos4.1.1/bin $path /usr/demo/SOUND /usr/local/x11r5/games /usr/e
tc)
source $HOME/.../.cshrc
umask 077
652 emily8>
```

2.9 Searching and Comparing Files

Searching for Text—You can use the **grep** command to search for a string in a text file. This is actually a very powerful utility, and you might want to learn more about it by typing **man grep**.

```
629 emily8> grep grep intro_unix.tex
{\bf Searching for Text}---You can use the {\bf grep} command to search for
you might want to learn more about how to use it by typing {\bf man grep}.

630 emily8> grep UNIX intro_unix.tex
\title{An Introduction To UNIX}
\section{Beginning UNIX}
you must use your UNIX account to log in to the computer---Emily1
added protection provided by the UNIX system so that curious eyes
{\bf The UNIX Manual}---On EML UNIX System the reference manual is online
(in the computer).  The manual contains descriptions of the UNIX commands online
so that you can refer to it on your screen as needed.  Unfortunately, you may
UNIX manuals and of the SUN workstation manuals are located in 616 Evans
\subsection{The UNIX File System}
{\bf How Is a File System Organized?}  UNIX files are organized into groups
EML UNIX computers are more complex than the fictional one depicted below.
{\bf Your Home Directory}---When you first login to a UNIX system, you are
{\bf Your Current Working Directory}---Whenever you are using a UNIX system,
can specify an absolute pathname by telling UNIX where the file is in relation
you can use a relative pathname.  That is, you can tell UNIX where the file
use UNIX's powerful output redirection to create a file.  See later sections
631 emily8>
```

Comparing Files—The **diff** and **cmp** commands allow you to compare two files to see if they are identical. The **diff** command displays the actual lines in each file that differ, whereas the **cmp** command lists the location in the first file where the first difference between them occurs. If the two files are identical, the prompt reappears. For more information type **man diff** or **man cmp**.

```
633 emily8> cp intro_unix.tex garbage
overwrite garbage? y
634 emily8> diff intro_unix.tex garbage
635 emily8> cmp intro_unix.tex garbage
636 emily8>
```

2.10 Handling File Security: File Access Permissions

UNIX Security—UNIX gives you the ability to protect your work as you see fit. You decide what you can

do to your work, and what others can do to it. The UNIX security system allows you some control over who can see, alter, and use your files and directories.

In the UNIX environment, those who may access a particular file or directory are divided into three classes, as shown in the following:

Class of User	Definition
owner	the user who owns the file/directory
group	the group of users who share access to a file. (Every user belongs to a group. In general, graduate students belong to the same group. To see what group you are in, type **groups**.)
other	all other users.

Each file or directory has a set of permissions that control access to it. For each class of users above, there are three permissions that may be granted in relation to each file or directory. These are shown here.

Permission	File	Directory
read(r)	Allows you to read the file	Allows you to list the contents of of the directory
write(w)	Allows you to create, delete, and modify the file.	Allows you to create and delete files in the directory, since this requires writing to the actual directory.
execute(x)	Allows you to run a file.	Allows you to list only the files whose names you specify, *not* the directory itself.

Finding Out What Permissions a File Has—When you create a file or a directory, it is assigned a default set of permissions, which you can change. With the command **ls -l**, you can see the permissions associated with a particular file as the first 10 characters on each line of the output. Be sure to use the correct absolute or relative pathname for the file.

```
671 emily8> ls -l
total 23
-rw-------  1 pinghua     20016 Jan 12 01:05 intro_unix.tex
-rw-r--r--  1 pinghua        21 Jan 12 00:35 test
drwxr-sr-x  2 pinghua       512 Jan 12 01:45 tex
drwxr-sr-x  2 pinghua       512 Aug  3 17:34 x11r5
672 emily8>
```

The first character indicates whether this file is a directory (**d**) or just a regular file (**-**). The next three groups consisting of three characters each (9 characters total) are the permissions granted for the owner, group, and other classes, respectively. A permission is granted if the letter is present, and not granted if the (−) is there instead.

Making Security Changes—You can change the permission modes using the **chmod** command. The plus (+) sign indicates access addition and minus (−) indicates access deletion. For more information type **man chmod** and **man umask**.

```
674 emily8> ls -l
total 23
-rw-------  1 pinghua     20016 Jan 12 01:05 intro_unix.tex
-rw-r--r--  1 pinghua        21 Jan 12 00:35 test
drwxr-sr-x  2 pinghua       512 Jan 12 01:45 tex
drwxr-sr-x  2 pinghua       512 Aug  3 17:34 x11r5
```

```
675 emily8> chmod go-r test
676 emily8> ls -l test
-rw-------  1 pinghua        21 Jan 12 00:35 test

677 emily8> chmod g+rw intro*
678 emily8> ls -l intro*
-rw-rw----  1 pinghua     20016 Jan 12 01:05 intro_unix.tex
679 emily8>
```

3 The Shell Program

About the Shell Program—As soon as you log in, the shell program is activated. Providing the interface between you and the system, the shell is the program that reads your commands and invokes other programs to carry them out. The default shell that you will be using is the **/bin/csh** or **C** shell. You can also change it to **/bin/tcsh** by using the **chsh** command. The shell program **/bin/tcsh** is an enhanced version of **/bin/csh**, allowing you to call back previous commands and edit them. For more information on changing your default shell program, type **man chsh**. [*Warning: do not change your default shell unless you really know what you are doing.*]

When the shell is ready for your command, it displays its prompt. In this document, we are using Pinghua's own prompt **680 emily8>** in the sample sessions. You can change your default prompt as shown later on in this section.

3.1 Customizing Your Session: Defining Your Shell Environment

UNIX allows you to customize your interaction with the shell in many ways. You can:

- create abbreviations for files or directories or lengthy commands;
- determine the directories the shell uses to search for the programs and commands to run;
- create the prompt the shell uses when it interacts with you, and
- have UNIX try to set your terminal type every time you dial in from home, without having to type the command yourself.

3.1.1 Shell Variables

Setting Shell Variables—Shell variables help determine how the shell program interacts with you. You can create your own shell variable and assign it a value with the **set** command. And you can use the variable in a command line by prefixing the variable name with $.

```
521 emily5> set workdir = /srv/accounts/projects/choices
522 emily5> echo $workdir
/srv/accounts/projects/choices

523 emily5> ls $workdir
News/           bin/            hlogit/         sst_docu/       workshop/
README          crane/          nag/            tar/            x/
```

```
524 emily5> unset workdir
525 emily5>
```

If you want to use the shell variable every time you interact with the **csh** or **tcsh** shell, add the command line to your **.cshrc** file. This is the **C Sh**ell **R**untime **C**ommands file, a dot file in your home directory.

Looking at Shell Variables—You can also use the **set** command to see shell variables and their values.

```
525 emily5> set
_etc    ()
_mpath  /usr/local/x11/man:/usr/local/ow3/man
_wpath  (/usr/local/x11/bin /usr/local/ow3/bin)
_x11    /usr/local/x11
addsuffix
argv    ()
cwd     /srv/accounts/grad/pinghua/pub/tex
echo_style      bsd
edit
filec
gid     130
history 500
home    /srv/accounts/grad/pinghua
hostname        emily5
ignoreeof
mail    (2 /usr/spool/mail/pinghua_young /etc/motd /usr/msgs)
mdir    /usr/spool/mail
noclober
notify
old     /srv/accounts/grad/pinghua
path    (/srv/accounts/grad/pinghua/bin /srv/accounts/projects/choices/bin
/srv/accounts/grad/pinghua/unix/mac /moby/pbmplus/bin
/tmp_mnt/moby/b/tiff/sun4_sunos4.1.1/bin
/usr/local/x11/bin /usr/local/ow3/bin /usr/local/bin /usr/lang /usr/ucb /bin /us
r/bin /usr/hosts . /usr/app/bin /usr/local/wp51/wpbin /usr/local/TeX/bin /usr/de
mo/SOUND /usr/local/x11r5/games /usr/etc)
prompt  ! emily5>
prompt2 loop:
prompt3 CORRECT>%R (y|n|e)?
savehist        500
shell   /bin/tcsh
shlvl   1
status  0
tcsh    6.02.00
term    xterm
tperiod 5
tty     ttyp1
uid     305
user    pinghua
version tcsh 6.02.00 (Cornell) 92/05/15 options 8b,nls,dl,al,dir
window_menu
526 emily5>
```

Some Important Shell Variables—A few shell variables have special meaning to the shell program. By assigning values to these variables, you customize the standard manner in which the shell executes your commands. Those covered here are: **path, prompt, history** and **term**.

About the Path Shell Variable—One of the most important shell variables is the path shell variable. The value for the path shell variable is a list of directories enclosed in parentheses. This list of directories is called your search path. UNIX searches these directories when it looks for a program that you specify in a command line. If a program is in your search path, you can type just the name of the program. Otherwise, you have to specify its absolute or relative pathname.

Suppose your search path is:

```
( /bin  /usr/local/x11r5/bin  /usr/bin )
```

Then whenever you type a program name or command, the shell looks first in the **/bin** directory, second in the directory **/usr/local/x11r5/bin** and finally in the directory **/usr/bin**. It keeps looking until it either finds the program or command with that name or finishes looking through all the directories in your search path. If it cannot find it among the directories listed in your search path, it's going to give you the error message of **Command not found**.

```
526 emily8> nslookup
nslookup: Command not found.
527 emily8> set path = (/usr/etc $path)
528 emily8> nslookup
Default Server:  econ.Berkeley.EDU
Address:  128.32.105.1

> exit
529 emily8>
```

The procedure shown in this example only adds one directory to your search path for your present interaction with the current shell. If you find that you often use a program or command not specified in your default search path, you can edit or add the command line **set path = (new_search_directory $path)** to your **.cshrc** file. To find out the value of your path shell variable, simply type **echo $path** at your shell prompt.

About the Prompt Shell Variable—The value for the prompt shell variable is what displays as the shell's prompt. You can customize your shell prompt by using the procedure shown in the following example.

```
532 emily8> set prompt="what now? "
what now? ls
intro_unix.tex          multi_ucblet.tex        short_loc_gid.tex
local_guide.tex         multi_ucblet_body.tex   ucblet.tex
what now?
```

In the tcsh, to change the prompt permanently, add the command line **set prompt = "your desired prompt"** in your **.cshrc** file.

About the History Shell Variable—The value associated with the history shell variable is the number of the commands last entered, which UNIX will save when you use the history command. See *Keeping Command History* later in Section 6. To set the number of commands that UNIX will save for subsequent sessions, add the command line **set history = 500** in your **.login** file (500 is just an example). In this case, UNIX will remember 500 commands for you, but only for the current work session. Once you logout, those 500 commands won't be remembered. To ask UNIX to keep the 500 commands from the previous session

available to you for the current session, add **set savehist = 500** in your **.cshrc** file. For more information read **man csh**.

About the Term Shell Variable—The value associated with the term shell variable is the type of terminal you are using. This is important when you are dialing in from home. If you are using vt100 or vt102 emulation mode with your PC or Mac communication software, then you probably don't have to set this variable explicitly. But if you are using some other emulation mode, then you have to set the terminal type explicitly to the type you are using. Otherwise, programs such as text editors won't work properly.

3.1.2 Modifying the .login and .cshrc Files

In general, you should be very careful when you start modifying these two files. Make sure you know what you are doing, and remember the line(s) you are adding. If it doesn't work, put # in front of the line so that it won't be executed. [Warning: do not blindly copy lines from someone else's dot files, or edit your dot files yourself unless you know exactly what you are doing. Otherwise you will come to grief. Additionally, never put another user's directory into a "set path" statement. Think about it: you'll be sorry when that account is closed.]

The difference between **.login** and **.cshrc** files is: when you login, UNIX executes the **.cshrc** file first and then **.login**. UNIX also executes your **.login** file once, whereas it executes your **.cshrc** file whenever you start a subshell.

After you add or modify command lines in the **.cshrc** or **.login** files, UNIX won't run the commands immediately. Unless you use the **source** command, you have to logout and then login in order for the changes to take effect.

```
532 emily8> source .login
533 emily8> source .cshrc
```

3.1.3 Environment Variables

These are similar to shell variables, but other programs besides the shell use their values. Typically, these environment variable names are all uppercase. An example of an environment variable is **PRINTER**. The value associated with this environment variable determines the default printer. The commands to set, unset, and look at environment variables are slightly different from those used with shell variables. Like shell variables, the command lines can be changed or added to your **.login** or **.cshrc** file.

```
537 emily8> setenv PRINTER lp1
538 emily8> unsetenv PRINTER lp1
539 emily8> printenv
HOME=/srv/accounts/grad/pinghua
SHELL=/bin/tcsh
TERM=vt100
USER=pinghua
PATH=/srv/accounts/grad/pinghua/bin:/srv/accounts/projects/choices/bin:
/srv/accounts/grad/pinghua/unix/mac:/usr/local/x11/bin:/usr/local/ow3/bin:
/usr/local/bin:/usr/lang:
/usr/ucb:/bin:/usr/bin:/usr/hosts:.:/usr/app/bin:/usr/local/wp51/wpbin:/usr/loca
l/TeX/bin:/usr/demo/SOUND:/usr/local/x11r5/games:/usr/etc
LOGNAME=pinghua
SHLVL=1
PWD=/srv/accounts/grad/pinghua/pub/tex
```

```
HOST=emily8.Berkeley.EDU
HOSTTYPE=sun4
WPTERM51=sunshell
LD_LIBRARY_PATH=/usr/local/x11/lib:/usr/local/ow3/lib
HELPPATH=/usr/local/x11/lib/help:/usr/local/ow3/lib/help
XFILESEARCHPATH=/usr/local/x11/lib/X11/%T/%N%C:/usr/local/x11/lib/X11/%T/%N:/usr
/local/ow3/lib/%T/%N%S
OPENWINHOME=/usr/local/ow3
MANPATH=/usr/local/x11/man:/usr/local/ow3/man:/usr/local/man:/usr/lang/man:/usr/
man:/usr/app/man
PAGER=more
EXINIT=set redraw
TERMINFO=/usr/share/lib/local/terminfo
TERMPATH=/usr/local/etc/termcap.local:/etc/termcap
GNUTERM=tek40
TERMCAP=d0|vt100|vt100-am:do=^J:co#80:li#36:cl=50\E[;H\E[2J:le=^H:bs:am:cm=5\E[%
i%d;%dH:nd=2\E[C:up=2\E[A:ce=3\E[K:cd=50\E[J:so=2\E[7m:se=2\E[m:us=2\E[4;1m:ue=2
\E[m:md=2\E[1m:mr=2\E[7m:mb=2\E[5m:me=2\E[m:is=\E[?7h\E[1;24r\E[24;1H:rf=/usr/li
b/tabset/vt100:rs=\E>\E[?3l\E[?4l\E[?5l\E[?7h\E[?8h:ks=\E[?1h\E=:ke=\E[?1l\E>:ku
=\EOA:kd=\EOB:kr=\EOC:kl=\EOD:kb=^H:ho=\E[H:k1=\EOP:k2=\EOQ:k3=\EOR:k4=\EOS:pt:s
r=5\EM:vt#3:xn:ti=\E[?7l:te=\E[?7h:sc=\E7:rc=\E8:cs=\E[%i%d;%dr:
MAIL=/usr/spool/mail/pinghua
VISUAL=/usr/ucb/vi
NAME=Pinghua Young
ORGANIZATION=University of California at Berkeley
540 emily8>
```

3.2 Creating Shorthand Names for Commands: Aliases

Aliases allow you to create shorthand names for frequently used or lengthy command lines, and to create modified versions of existing system commands ("customizing" commands). When the alias appears in the command line that the shell reads, its text is replaced by the definition of the alias. You use the **alias** command to create aliases and the **unalias** command to delete them. To display a definition of an alias, just type **alias** and the name.

```
541 emily8> alias a alias
542 emily8> a f finger
543 emily8> a bai f jushan@athena.mit.edu
544 emily8> bai
[athena.mit.edu]
Login name: jbai                        In real life: Jushan Bai
Nickname:                               Home phone:
Office: E52-274B                        Office phone: 617-253-6217
Electronic mail address: jbai@ATHENA.MIT.EDU

545 emily8> alias bai
finger jushan@athena.mit.edu
546 emily8>
```

Arguments in Aliases—Unless you specify otherwise, the C shell assumes that all arguments come at the end of the alias definition. For example, suppose **cx** is an alias for **chmod +x**. Then **cx *** would expand to **chmod +x ***.

What if we want arguments placed somewhere besides at the end of an alias definition? For example, let's take a look the **find** command. To locate the file **intro_unix.tex** starting from our home directory, we type

```
521 cox> find ~ -name intro_unix.tex -print
/srv/accounts/grad/pinghua/pub/tex/intro_unix.tex
522 cox>
```

We want to create an alias called **loc** that takes a filename as an argument to the **find** command. A way to do this is

```
alias loc 'find ~ -name \!* -print'
```

To avoid accidental removal of files, we strongly suggest that you put the following aliases in your **.cshrc file**:

```
alias rm rm -i
alias cp cp -i
alias mv mv -i
```

You might also want to add **alias ls ls -F** to your **.cshrc** file.

4 Input/Output Redirection and Pipes

4.1 Sending Results Where You Want Them: Output Redirection

Many UNIX commands, such as **who, ls**, and **date**, normally display their output on the terminal screen. Once it is overwritten or scrolls off the screen, you must execute the command again to view it (unless, of course, you are using a windows system that allows you to scroll back). Output redirection lets you save output in a file for later reference or for use as the input to another program.

Standard Output—We refer to the output that normally appears on the screen (excluding error messages) as *standard output*. We may send standard output to a file using output redirection. For example, The command **who** > **users.list** saves the output from **who** in a file called **users.list**.

```
532 cox> ls users.list
users.list not found
533 cox> who > users.list
534 cox> cat users.list
mcfadden console Jan 13 09:47
goldman  ttyp0   Dec 11 11:52
goldman  ttyp1   Dec 14 09:17
goldman  ttyp2   Nov 23 15:19
goldman  ttyp3   Dec  1 09:02
goldman  ttyp4   Jan 12 10:53
goldman  ttyp5   Jan  4 09:08
goldman  ttyp6   Jan  4 11:36
mcfadden ttyp7   Jan 13 09:48
mcfadden ttyp8   Jan 13 09:48
goldman  ttyp9   Jan 15 12:01
goldman  ttypa   Jan 14 11:56
```

```
mcfadden ttypb   Jan 14 15:14
pinghua  ttypc   Jan 17 17:21   (econnet.Berkeley)
535 cox>
```

We may say that > redirects the output. Notice that in the above sample session, the file **users.list** did not exist. The C shell simply created it and put **who**'s output there. If the file **users.list** had existed before we issued **who > users.list** command, its previous contents would have been destroyed. If you don't want this to happen, use >> instead of >. For example,

```
536 cox> ps -uax | grep pinghua >> users.list
537 cox> cat users.list
mcfadden console Jan 13 09:47
goldman  ttyp0   Dec 11 11:52
goldman  ttyp1   Dec 14 09:17
goldman  ttyp2   Nov 23 15:19
goldman  ttyp3   Dec  1 09:02
goldman  ttyp4   Jan 12 10:53
goldman  ttyp5   Jan  4 09:08
goldman  ttyp6   Jan  4 11:36
mcfadden ttyp7   Jan 13 09:48
mcfadden ttyp8   Jan 13 09:48
goldman  ttyp9   Jan 15 12:01
goldman  ttypa   Jan 14 11:56
mcfadden ttypb   Jan 14 15:14
pinghua  ttypc   Jan 17 17:21   (econnet.Berkeley)
pinghua   2934  0.0  0.3   32  208 pc S    18:00   0:00 grep pinghua
pinghua   2831  0.0  0.8  360  524 pc S    17:21   0:02 -tcsh (tcsh)
pinghua   2892  0.0  0.0  184    0 pc TW   17:35   0:03 vi intro_unix.tex
pinghua   2933  0.0  0.7  200  436 pc R    18:00   0:00 ps -uax
538 cox>
```

In the above sample session, the output from **ps -aux | grep pinghua** is appended to the file **users.list**.

Safer Redirection and Appending—Redirection can easily destroy a valuable file accidentally. To avoid this, the C shell provides a protective version of redirection and appending. This works as follows:

- It does not allow redirection if the file already exists.
- It does not allow appending if the file does not already exist.

This keeps you from accidentally destroying a valuable file or creating a new file just because of absent-mindedness or typing errors. The command **set noclobber** activates this additional protection.

```
539 cox> set noclobber
540 cox> who > users.list
users.list: File exists.
541 cox> date >> status
status: No such file or directory.
542 cox>
```

You may want to put **set noclobber** in your **.cshrc** file. (Note: in the EML system, noclobber" is already set in the system default standard .cshrc file for all accounts.) To overwrite this extra protection, use >! and >>! instead of > and >> when redirecting and appending standard output.

4.2 Getting the Input You Need: Input Redirection

You can redirect input as well as output. For example, the **mail** command normally reads input from your terminal as shown in the sample session below.

```
546 cox> mail pinghua@econ
Subject: test
This is a test.  Please ignore.
.
Cc:
547 cox>
```

But in practice, you normally want to create letters with an editor that allows you to revise them, reformat them, correct the spelling, and remove caustic remarks and other things you probably shouldn't have included anyway. The command **mail pinghua@econ < intro_unix.tex** mails Pinghua's file **intro_unix.tex** to himself.

```
549 cox> mail pinghua@econ < intro_unix.tex
550 cox>
```

Input redirection is particularly convenient with commands such as **mail** and **tr** (translate) that only read from the standard input. For example, suppose we have a file called **report** that contains all uppercase characters, and we want to create a version in lowercase. The following command converts all uppercase letters to lowercase in the file **report** and stores the result in the file **report.low**.

```
551 cox> tr "[A-Z]" "[a-z]" < report > report.low
552 cox>
```

4.3 Making the Right Connections: Pipes

Often it's convenient to make the output of one command become the input of another command without saving any temporary files inbetween. The UNIX C Shell provides a means of connecting the output and input of two commands via a mechanism called a *pipe*. The pipe character | is placed between two commands when the first command's output should be the input of the second.

```
556 cox> ypcat passwd | awk -F: '{print $1, $2}' | grep pinghua
pinghua DTWrhLO2gb7UI
557 cox> ypcat group | grep pinghua
choices:*:601:mcfadden,katagiri,pinghua,ruud,train
558 cox>
```

5 Job Control

Because UNIX is **multi-tasking**, you can run more than one process or job at a time. If a job is actively communicating with your terminal, it is running in the **foreground**. If you have started other jobs that are still running but not actively communicating with your terminal, they are in the **background**.

5.1 Foreground Jobs

Normally, when you begin a job by typing a command at the prompt (without ending it with an &), it runs in the foreground. With a foreground job, we simply wait until it finishes. The C shell displays its prompt when it is ready for our next command line. This works fine for short jobs. If the waiting time is excessive, we can simply interrupt the job (usually by typing **Control–C**), and the C shell will terminate it prematurely. But what if the job is a long one?

5.2 Background Jobs

The C shell lets you assign long jobs to the background while you proceed with other tasks. Of course, this works best when the background jobs needs little or no attention. To make a command execute in the background, put an ampersand (&) after it.

```
512 cox> delatex intro_unix.tex | spell > misspell.txt &
[2] 4185 4186
513 cox>
```

When we make the command run in the background, the C shell responds with a job number (inside brackets) and the process identifications (**4185** for **delatex** and **4186** for **spell**). It then immediately displays the prompt, and we can enter more commands.

5.3 Controlling Jobs

To control and monitor background execution, we need commands that operate on jobs and processes. Two important commands, **jobs** and **ps**, give us status information on background jobs. The **ps** command displays a table of all your currently executing processes.

```
519 cox> ps
  PID TT STAT  TIME COMMAND
 4130 pc S     0:01 -tcsh (tcsh)
 4167 pc TW    0:02 vi intro_unix.tex
 4212 pc T     0:00 ftp sumex.stanford.edu
 4213 pc T     0:00 rlogin toto -l pinghua
 4214 pc T     0:00 rlogin toto -l pinghua
 4215 pc R     0:00 ps
520 cox> ps -aux | grep pinghua
pinghua   4216 38.5  0.7  200  436 pc R    13:56   0:00 ps -aux
pinghua   4218  7.7  0.3   32  208 pc S    13:56   0:00 grep pinghua
pinghua   4214  0.0  0.0   48   24 pc T    13:55   0:00 rlogin toto -l pinghua
pinghua   4130  0.0  0.8  360  504 pc S    13:31   0:01 -tcsh (tcsh)
pinghua   4213  0.0  0.4   48  280 pc T    13:55   0:00 rlogin toto -l pinghua
pinghua   4167  0.0  0.0  188    0 pc TW   13:35   0:02 vi intro_unix.tex
pinghua   4212  0.0  0.0   84    0 pc TW   13:55   0:00 ftp sumex.stanford.edu
521 cox> jobs
[1]    Suspended              vi intro_unix.tex
[2]  - Suspended              ftp sumex.stanford.edu
[3]  + Suspended (tty output) rlogin toto -l pinghua
522 cox>
```

The **jobs** command labels each job by number (in brackets). A plus sign indicates current background job. A minus sign indicates the next job in line. All other jobs are unmarked. The only significance of the current job is that you can use shorthand notation to refer to it in job control commands.

5.4 Stopping a Foreground Job

You can stop a job running in the foreground before it's finished by holding down **CTRL** and typing **Z**. At this point you return to the shell where you started the job. The job has stopped altogether.

```
529 cox> archie -s macgs > ghostscript
^Z
Suspended
530 cox> fg
archie -s macgs > ghostscript
^Z
Suspended
531 cox> jobs
[2]    Suspended              ftp sumex.stanford.edu
[3]  - Suspended              vi intro_unix.tex
[4]  + Suspended              archie -s macgs > ghostscript
532 cox>
```

5.5 Continuing and Moving Jobs

To go back and continue a stopped job, use the **bg** command. The command **bg** with no argument will simply continue the most recently stopped job (the one with a plus sign) in the background where it left off. If you want to continue a specific job you can type **bg %3** where **3** is a job number. To move a background job to foreground, use **fg** command. To bring job number 5 from background to foreground, for example, use the **fg %5**.

```
541 cox> jobs
[2]    Suspended              ftp sumex.stanford.edu
[3]  - Suspended              vi intro_unix.tex
[4]  + Suspended (signal)     archie -s macgs > ghostscript
542 cox> bg %4
[4]    archie -s macgs > ghostscript &
543 cox> fg %2
ftp sumex.stanford.edu

ftp> ^Z
Suspended
544 cox> jobs
[2]  + Suspended              ftp sumex.stanford.edu
[3]  - Suspended              vi intro_unix.tex
[4]    Running                archie -s macgs > ghostscript
545 cox>
```

5.6 Stopping a Background Job

To stop a background running job, use the **stop** command. For example, to stop background job number 4, use **stop %4**. To continue a stopped background job, use **bg** as illustrated previously.

```
546 cox> jobs
[2]  - Suspended              ftp sumex.stanford.edu
[3]  + Suspended              vi intro_unix.tex
[4]    Running                archie -s macgs > ghostscript
547 cox> stop %4

[4]  + Suspended (signal)     archie -s macgs > ghostscript
548 cox> bg %4
[4]    archie -s macgs > ghostscript &
549 cox>
```

If you still have stopped jobs when you logout, a reminder message will appear on your screen. Although it is not recommended, you can choose to ignore the message and type **logout** again. Instead, you should use the **jobs** command to review the suspended jobs and then either use the **bg** command to continue them or the **kill** command to terminate them altogether.

5.7 Killing Jobs and Processes

To terminate a job or process running in the background when you are sure you don't want it to finish, use the **kill** command. This is especially important when you are running a big job or process that freezes your screen. If this happens, go to another console and login there. Then remote login (using **rlogin**) to the frozen machine, use the **ps** command to find the process that caused the trouble, and kill it.

```
554 cox> jobs
[2]  - Suspended              ftp sumex.stanford.edu
[3]  + Suspended              vi intro_unix.tex
[4]    Running                archie -s macgs > ghostscript
555 cox> kill -9 %2

[2]    Killed                 ftp sumex.stanford.edu
556 cox> ps
  PID TT STAT  TIME COMMAND
 4130 pc S     0:03 -tcsh (tcsh)
 4220 pc T     0:05 vi intro_unix.tex
 4229 pc IW    0:00 archie -s macgs
 4264 pc R     0:00 ps
557 cox> kill -9 4229

[4]    Killed                 archie -s macgs > ghostscript
558 cox>
```

6 Keeping Command History

6.1 The History List

The history mechanism saves your commands in a list, thus allowing you to avoid repetitive typing. You can use the list to correct a spelling error in a long command, recall a filename, or even repeat an entire command. You can also build new commands from old ones.

To display the history list, use the **history** command (this is aliased to just "h" on the EML system). To display only a partial list, use **history 5**, which displays the last five commands. Note that **history | tail -5** will give you the same result.

```
571 cox> history 5
   567  14:38   rm mis*
   568  14:38   ls
   569  14:38   ckall
   570  14:39   fg
   571  14:45   history 5
572 cox>
```

6.2 Reissuing Past Commands

The C shell refers to past commands as *events*. Since the history list lets you refer to events by number, you may want the current command number displayed with the prompt. Putting an exclamation point in the prompt string does this. For example, the command

```
set prompt = '\! `hostname`\% '
```

changes the prompt to the command number followed by the host name and then a percent sign.

Most history commands begin with **!**. For example, typing **!!** reissues the previous command. To reexecute other commands, type **!** and the event number.

```
580 cox> history 5
   576  14:54   history 5
   577  14:54   latex intro_unix
   578  14:54   latex intro_unix
   579  14:54   fg
   580  14:55   history 5
581 cox> !577
latex intro_unix
This is TeX, C Version 3.141
(intro_unix.tex
LaTeX Version 2.09 <25 March 1992>
(/usr/local/TeX/tex/macros/article.sty
Standard Document Style `article' <14 Jan 92>.
(/usr/local/TeX/tex/macros/art10.sty))
(/srv/accounts/grad/pinghua/tex/inputs/fullpage.sty) (intro_unix.aux) [1]
[2] [3] [4] [5] [6] [7] [8] [9] [10] [11] [12] [13] [14] [15] [16] [17]
[18] [19] [20] [21] [22] [23] [24] [25] (intro_unix.aux) )
(see the transcript file for additional information)
Output written on intro_unix.dvi (25 pages, 63460 bytes).
```

```
Transcript written on intro_unix.log.
582 cox>
```

The C shell also allows you to find an event by matching the initial string. For example, typing **!vi** finds the most recent command starting with **vi**. We can also search for events by matching a string anywhere in the past command line. All we must do is put question marks around the target string. For example, the command **!?intro?** searches for and executes the most recent command containing the string **intro**.

```
588 cox> !?arch?
archie -s macgs > ghostscript
^Z
Suspended
589 cox>
```

6.3 Building New Commands

You can build new commands by selecting words from previous commands. **!$** refers to the last word of the last command. In fact, we can use any word from any event on the history list. The C shell numbers the words in each command, starting at 0.

```
593 cox> !?la?
latex intro_unix
This is TeX, C Version 3.141
(intro_unix.tex
LaTeX Version 2.09 <25 March 1992>
(/usr/local/TeX/tex/macros/article.sty
Standard Document Style `article' <14 Jan 92>.
(/usr/local/TeX/tex/macros/art10.sty))
(/srv/accounts/grad/pinghua/tex/inputs/fullpage.sty) (intro_unix.aux) [1]
[2] [3] [4] [5] [6] [7] [8] [9] [10] [11] [12] [13] [14] [15] [16] [17]
[18] [19] [20] [21] [22] [23] [24] [25] [26] (intro_unix.aux) )
(see the transcript file for additional information)
Output written on intro_unix.dvi (26 pages, 65320 bytes).
Transcript written on intro_unix.log.

594 cox> dvips -o !$.ps !$.dvi
dvips -o intro_unix.ps intro_unix.dvi
This is dvips 5.495 Copyright 1986, 1992 Radical Eye Software
' TeX output 1993.01.18:1505' -> intro_unix.ps
<tex.pro>. [1] [2] [3] [4] [5] [6] [7] [8] [9] [10] [11] [12] [13] [14] [15]
[16] [17] [18] [19] [20] [21] [22] [23] [24] [25] [26]
595 cox>
```

6.4 Editing Previous Commands

The C shell provides a very limited mechanism for editing previous commands. To correct a single mistake in the previous command, use ^ (caret). You can also use **s** or **gs** as illustrated in the following sample session.

```
600 cox> rm ghostscriot
rm: ghostscriot: No such file or directory
```

```
601 cox> ^ot^pt
rm ghostscript
rm: remove ghostscript? y

602 cox> dvips -o intro_unx.ps intro_unx.dvi
This is dvips 5.495 Copyright 1986, 1992 Radical Eye Software
dvips: ! DVI file can't be opened.
603 cox> !!:gs/unx/unix
dvips -o intro_unix.ps intro_unix.dvi
This is dvips 5.495 Copyright 1986, 1992 Radical Eye Software
' TeX output 1993.01.18:1505' -> intro_unix.ps
<tex.pro>. [1] [2] [3] [4] [5] [6] [7] [8] [9] [10] [11] [12] [13] [14] [15]
[16] [17] [18] [19] [20] [21] [22] [23] [24] [25] [26]
604 cox> !!:s/unix/unx
dvips -o intro_unx.ps intro_unix.dvi
This is dvips 5.495 Copyright 1986, 1992 Radical Eye Software
' TeX output 1993.01.18:1505' -> intro_unx.ps
<tex.pro>. [1] [2] [3] [4] [5] [6] [7] [8] [9] [10] [11] [12] [13] [14] [15]
[16] [17] [18] [19] [20] [21] [22] [23] [24] [25] [26]
605 cox>
```

If you find yourself constantly editing previous commands, then you probably should consider changing your default shell from **/bin/csh** to **/bin/tcsh**. To try out tcsh, simply type **tcsh** at your shell prompt (type **exit** to get out of tcsh and back to the C shell). Once you are in tcsh, use **Control**–**P** to recall previous commands and **Control**–**N** to go to next command. Within a command line, use **Control**–**B** and **Control**–**F** to move backward and forward. Use **Control**–**D** to overstrike and **delete** or **backspace** key to delete a character. All these commands are the same as those of **emacs**.