UNIX SHELL REVIEW
AND
C REVIEW PT. 1

David S. Wise, Chris Haynes, Jeff Whitmer

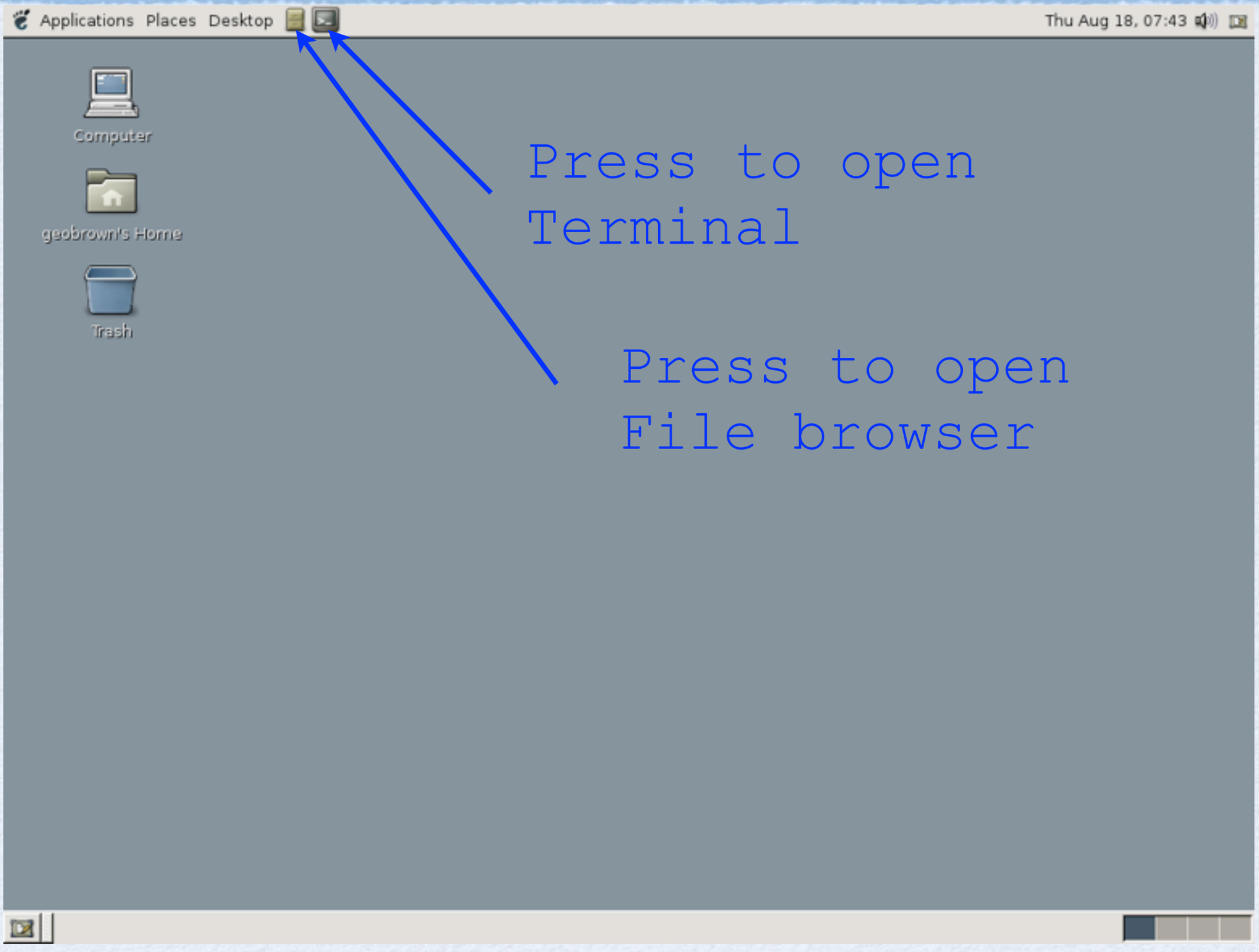
Bryce Himebaugh

Computer Structures Fall 2013

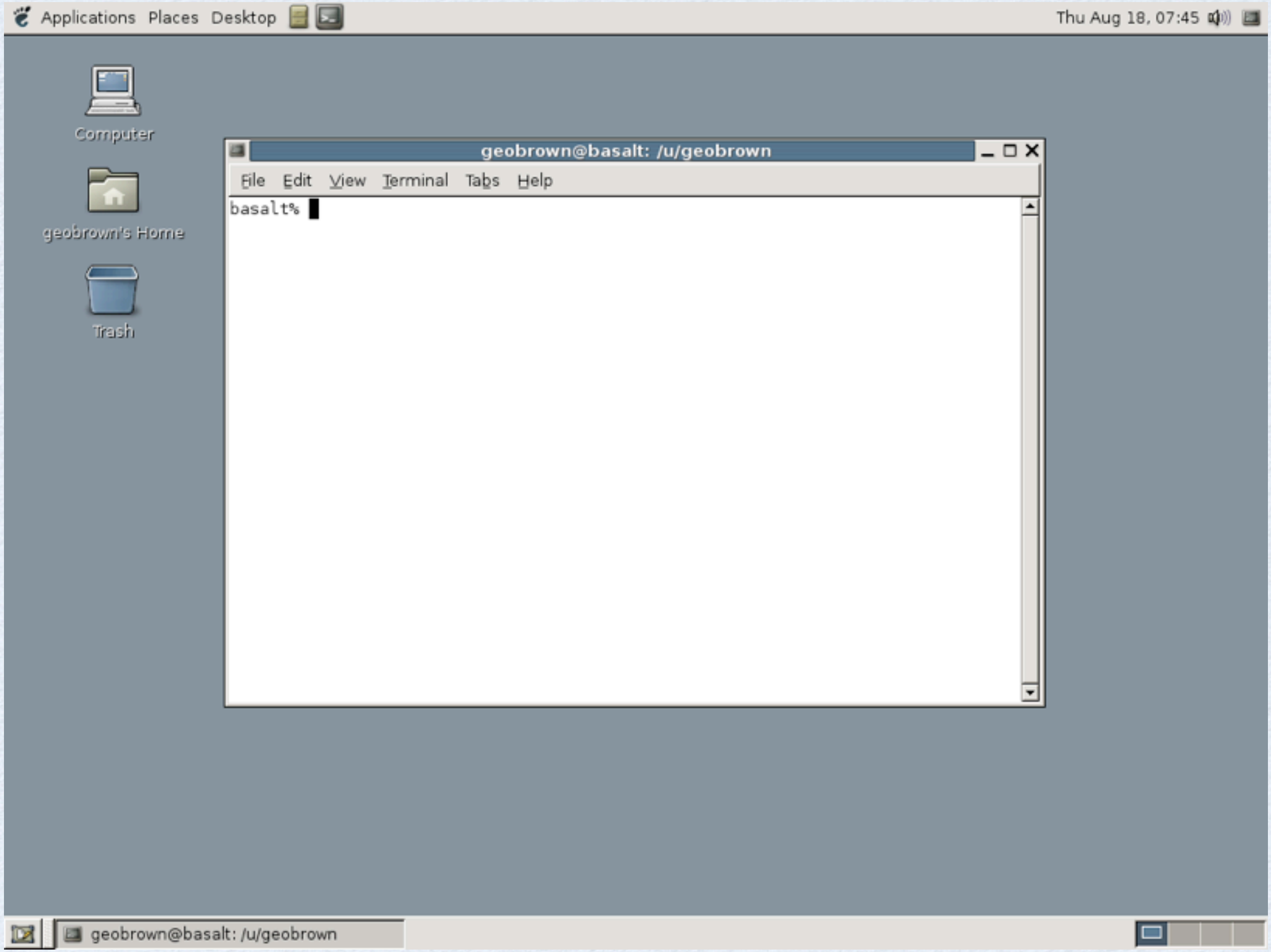
OVERVIEW

- Intro to the lab host environment: Burrow Unix system
- Unix shell (bash)
  - basic commands
  - how to find out more
- Unix files and directories

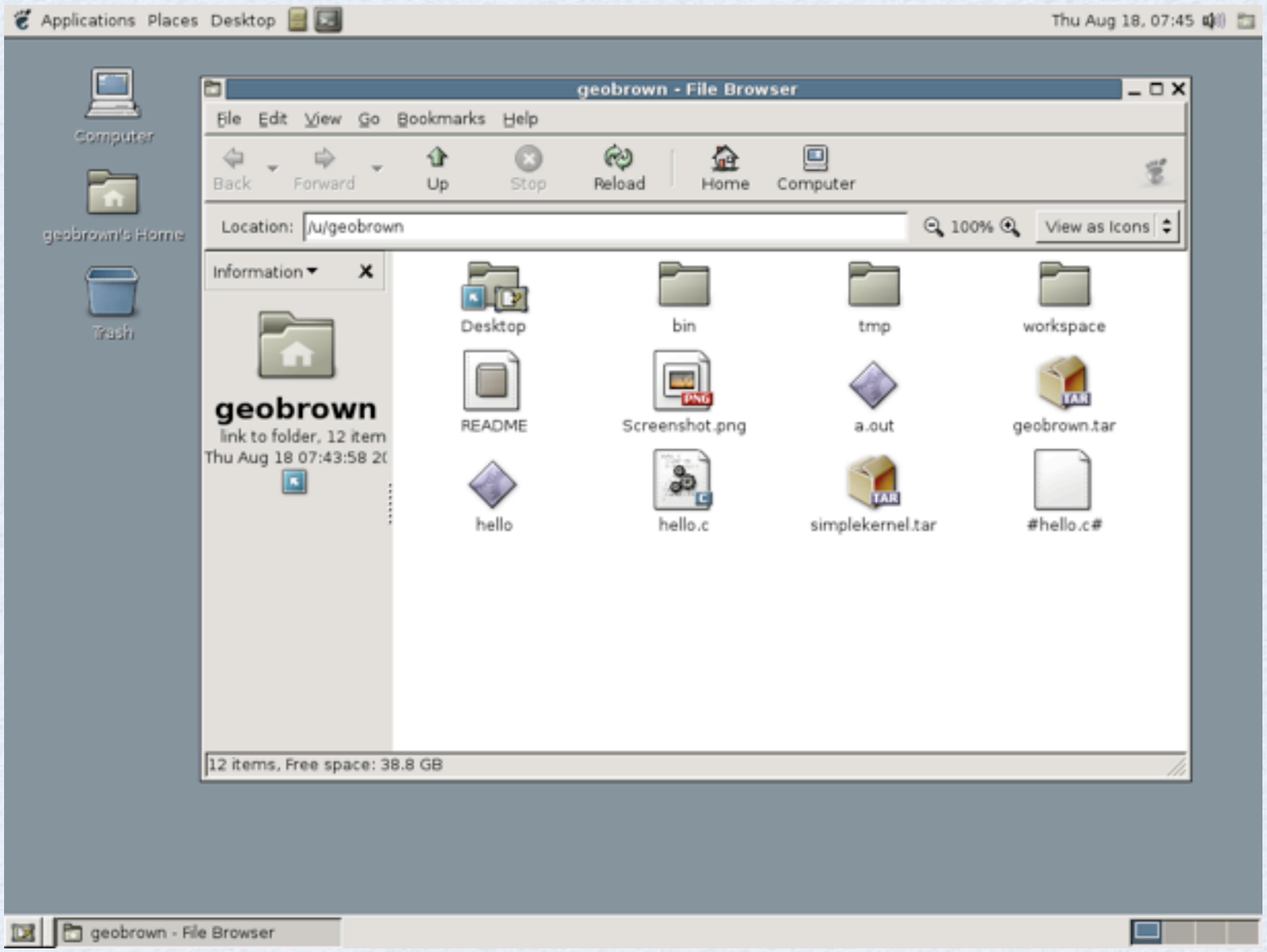
GNOME UNIX DESKTOP



TERMINAL

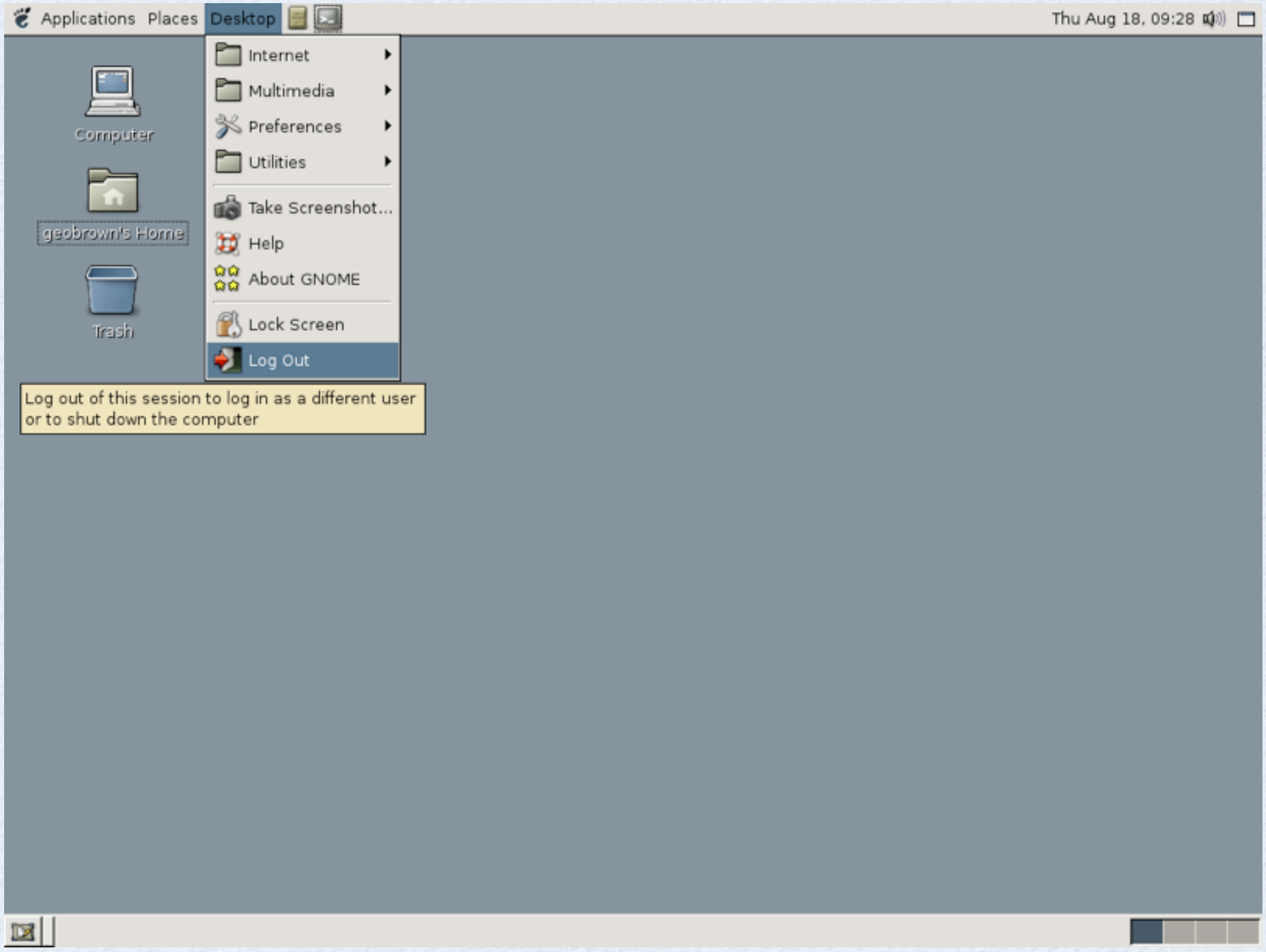

Applications
Places
Desktop
Thu Aug 18, 07:45
Computer
geobrown's Home
Trash
geobrown@basalt: /u/geobrown
File Edit View Terminal Tabs Help
basalt%
geobrown@basalt: /u/geobrown

FILE BROWSER

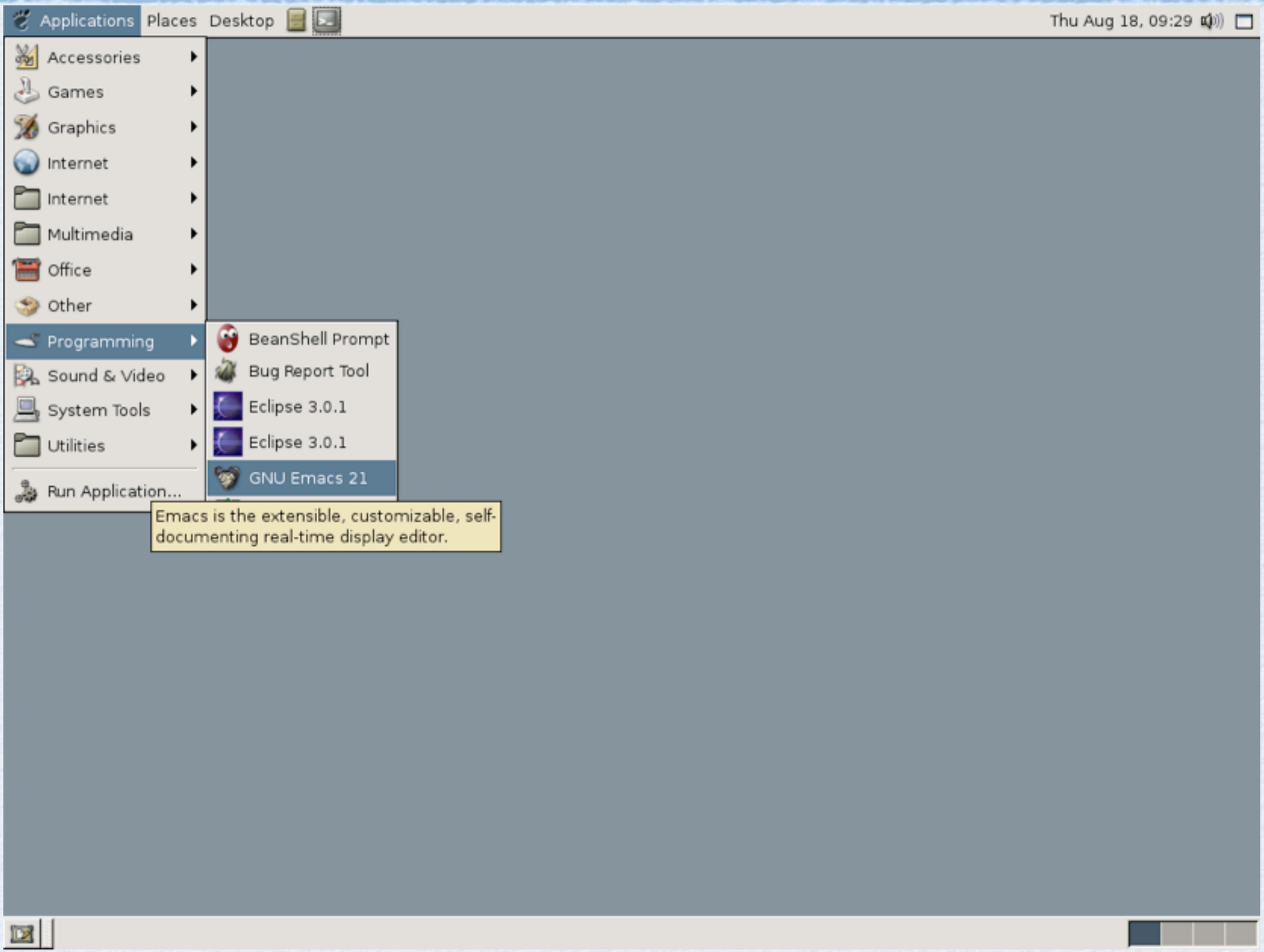


Learn to use shell commands in a terminal window instead of the file browser most of the time.

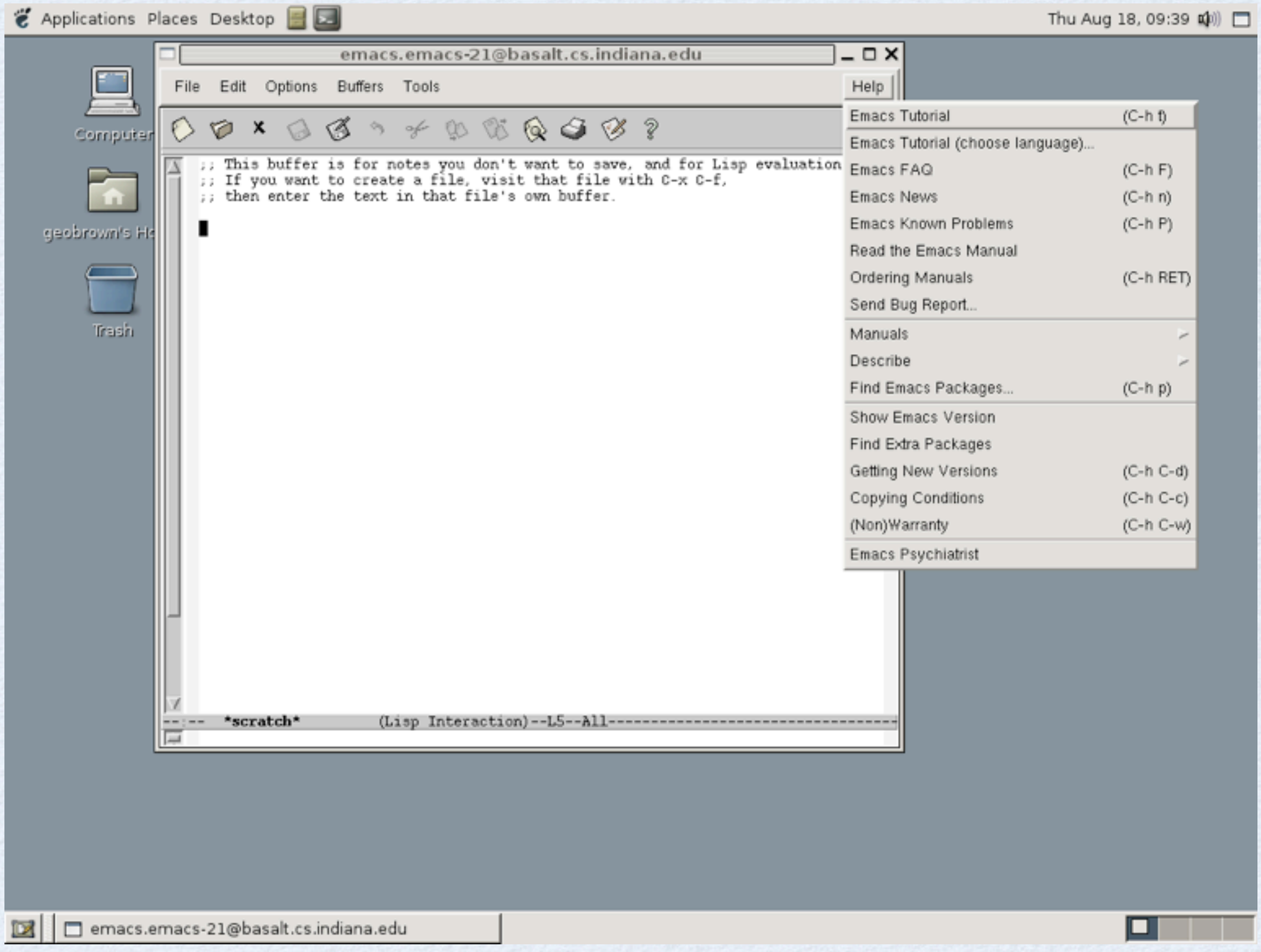
Log out


Applications
Places
Desktop
Thu Aug 18, 09:28
Computer
geobrown's Home
Trash
Internet
Multimedia
Preferences
Utilities
Take Screenshot...
Help
About GNOME
Lock Screen
Log Out
Log out of this session to log in as a different user or to shut down the computer

EMACS EDITOR



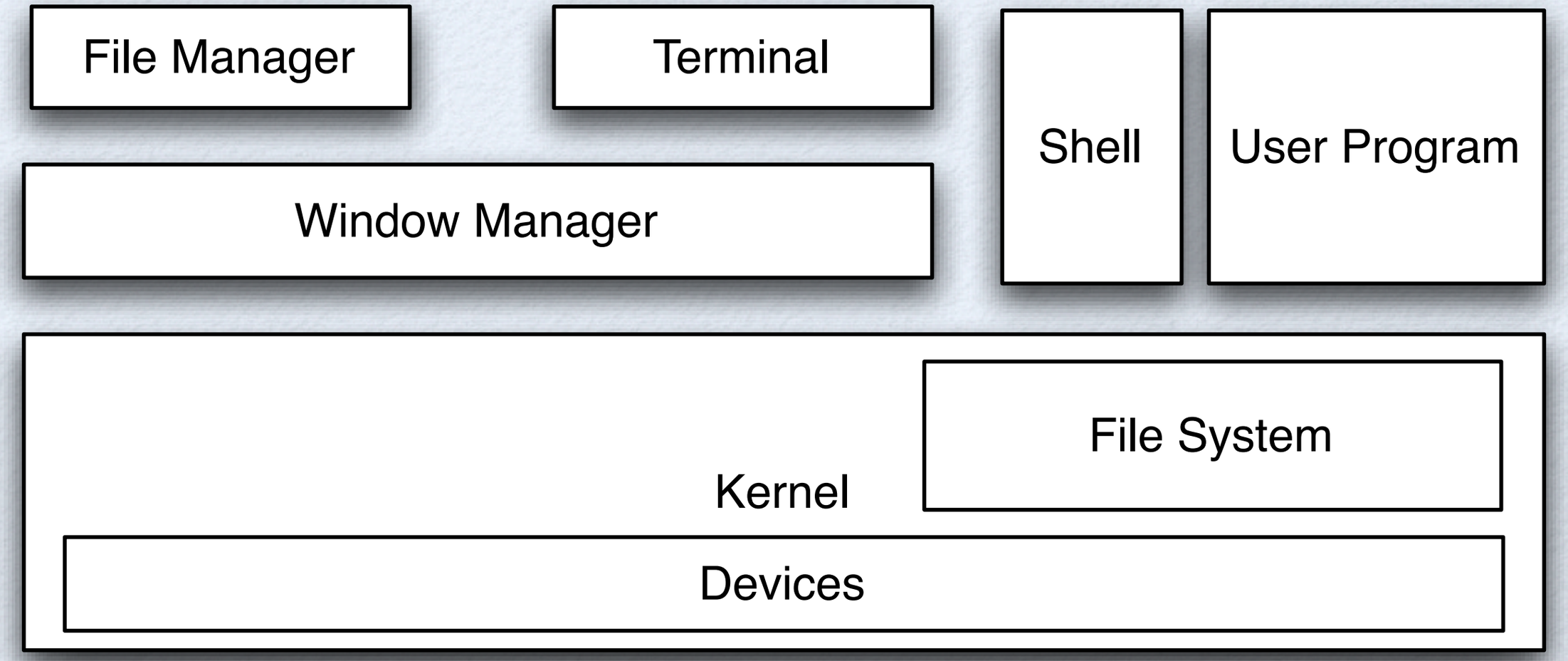
EMACS TUTORIAL



REMOTE ACCESS

- You can open a shell on a burrow machine from another machine.
  - use the SSH (secure shell) protocol with an SSH GUI or via the following command in another shell:
    - `ssh <username>@silo.soic.indiana.edu`
- This will not work when a hardware (USB) connection to the Discovery board is required.

OPERATING SYSTEM STRUCTURE



UNIX COMMANDS

The most versatile interactive way to communicate with Unix is through a **command-line interface** called a **shell**, running in a **terminal** window.

```
$ date
Fri Aug 29 09:32:32 CDT 2003
$
```

The **shell prompt** is customizable. Here it is **$**.

Tutorials: http://www.december.com/unix/tutor/

WHAT? WHERE? WHICH?

```
$ whatis whatis
whatis   (1)  - search the whatis database for complete words
$ whatis man
man                  (1)  - format and display the on-line manual pages
man                  (1p)  - display system documentation
man                  (7)  - macros to format man pages
man                  (rpm) - A set of documentation tools: man, apropos
and whatis.
man-pages            (rpm) - Man (manual) pages from the Linux
Documentation Project.
man.config [man] (5)  - configuration data for man
$ whereis man
man: /usr/bin/man /etc/man.config /usr/local/man /usr/share/man /usr/
share/man/man7/man.7.gz /usr/share/man/man1/man.1.gz /usr/share/man/
man1p/man.1p.gz
$ which man
/usr/bin/man
```

ANATOMY OF COMMAND-- OPTIONS

```
$ uname -s
Linux
$ uname -m
i686
$ uname --machine
i686
$ uname
Linux
```

ANATOMY OF COMMAND -- ARGUMENTS

```
$ date
Fri Aug 29 09:36:55 CDT 2003
$ cal
     August 2003
Su Mo Tu We Th Fr Sa
                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 27 28 29 30
31
$ cal 02 2001
   February 2001
 S  M Tu  W Th  F  S
              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 27 28
$
```

ANATOMY OF COMMAND -- FILE NAMES

```
$ ls
apple.txt  applebad.txt  applegood.txt bag.txt b.txt fish
$ ls *.txt
apple.txt  applebad.txt  applegood.txt bag.txt b.txt
$ ls a*
apple.txt  applebad.txt  applegood.txt
$ ls b??.txt
bag.txt
$
```

LS COMMAND

```
$ ls
apple.txt  applebad.txt  applegood.txt bag.txt basket.txt
$ ls -a
./              apple.txt       applegood.txt  basket.txt
../             applebad.txt    bag.txt
$ ls -l
total 5
-rw-r--r-- 1 december december  21 Dec 19 21:19 apple.txt
-rw-r--r-- 1 december december  21 Dec 19 21:59 applebad.txt
-rw-r--r-- 1 december december  23 Dec 19 21:41 applegood.txt
-rw-r--r-- 1 december december  21 Dec 19 21:38 bag.txt
-rw-r--r-- 1 december december  91 Dec 19 21:19 basket.txt
$ ls -s
total 5
   1 apple.txt          1 applegood.txt     1 basket.txt
   1 applebad.txt       1 bag.txt
$ ls -lst
total 5
   1 -rw-r--r-- 1 december december  21 Dec 19 21:59 applebad.txt
   1 -rw-r--r-- 1 december december  23 Dec 19 21:41 applegood.txt
   1 -rw-r--r-- 1 december december  21 Dec 19 21:38 bag.txt
   1 -rw-r--r-- 1 december december  91 Dec 19 21:19 basket.txt
   1 -rw-r--r-- 1 december december  21 Dec 19 21:19 apple.txt
$
```

ONLINE HELP

You can get online information on almost any Unix command, by using **man**. For example, to get information on all the options available for the **ls** command (there is a very large number of them), type

```
man ls
```

and a detailed (though terse) description of everything **ls** does will then appear on the screen. By the way, this description will be automatically piped through **more**, so as usual, just hit the space bar when you are ready to go to another screenfull.

At the end of the output, there will also be pointers to other commands related to the one requested, as well as lists of any files used by the command, such as "startup" files via which the command will have certain options set before execution.

Bare Minimum Unix introduction:
http://heather.cs.ucdavis.edu/~matloff/UnixAndC/Unix/UnixBareMn.pdf

REDIRECTING OUTPUT

In Unix, you can **redirect** standard output to go to a file. This output redirection is very useful if you want to save the output of a command to a file rather than just letting it flash across the screen and eventually scroll away from view.

To do so use the > sign after the command and before a file name. For example, you can redirect the output of the date command to a file called apple:

```
$ date > apple
$
$ cat apple
Tue Dec 19 21:16:43 CST 2000
$ date > apple
$ cat apple
Tue Dec 19 21:17:07 CST 2000
$
```

APPENDING OUTPUT TO FILE

> overwrites output, we can also append

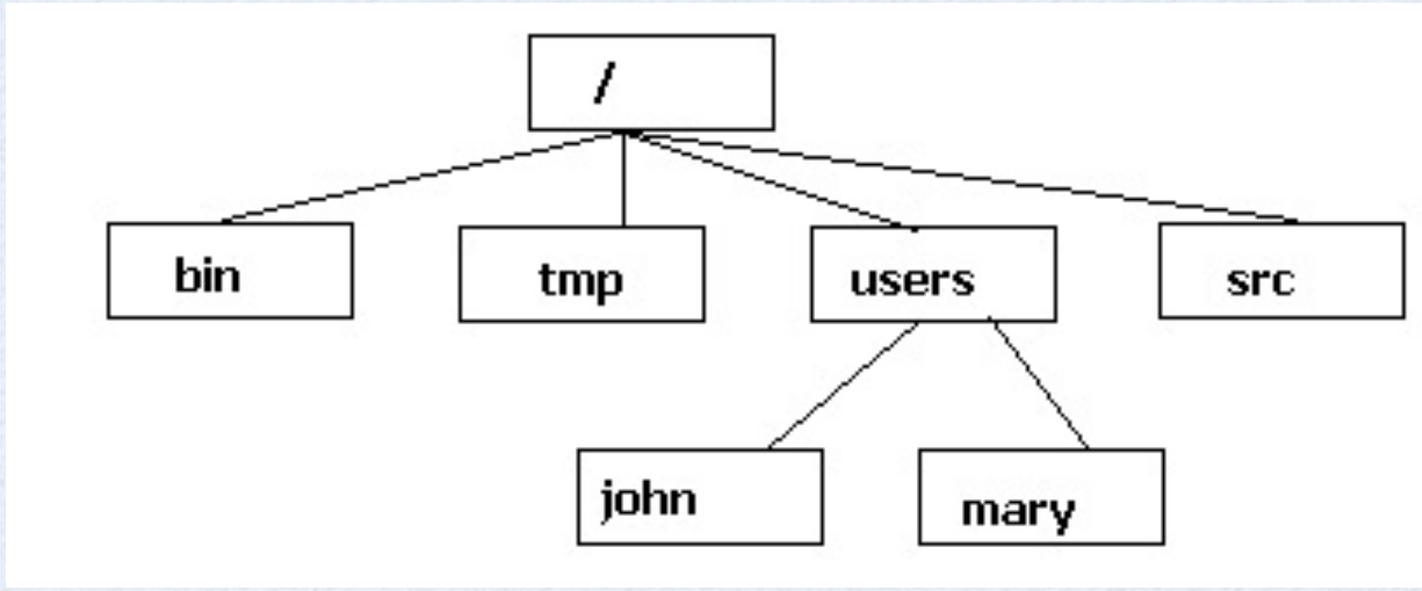
```
$ cat apple
Tue Dec 19 21:17:07 CST 2000
$ date >> apple
$ cat apple
Tue Dec 19 21:17:07 CST 2000
Tue Dec 19 21:17:53 CST 2000
$
```

NAVIGATING FILE SYSTEM

```
$ pwd
/users/john
$
```

```
$ pwd
/users/john
$ cd ..
$ pwd
/users
$
```

```
$ pwd
/users
$ cd ..
$ pwd
/
$
```

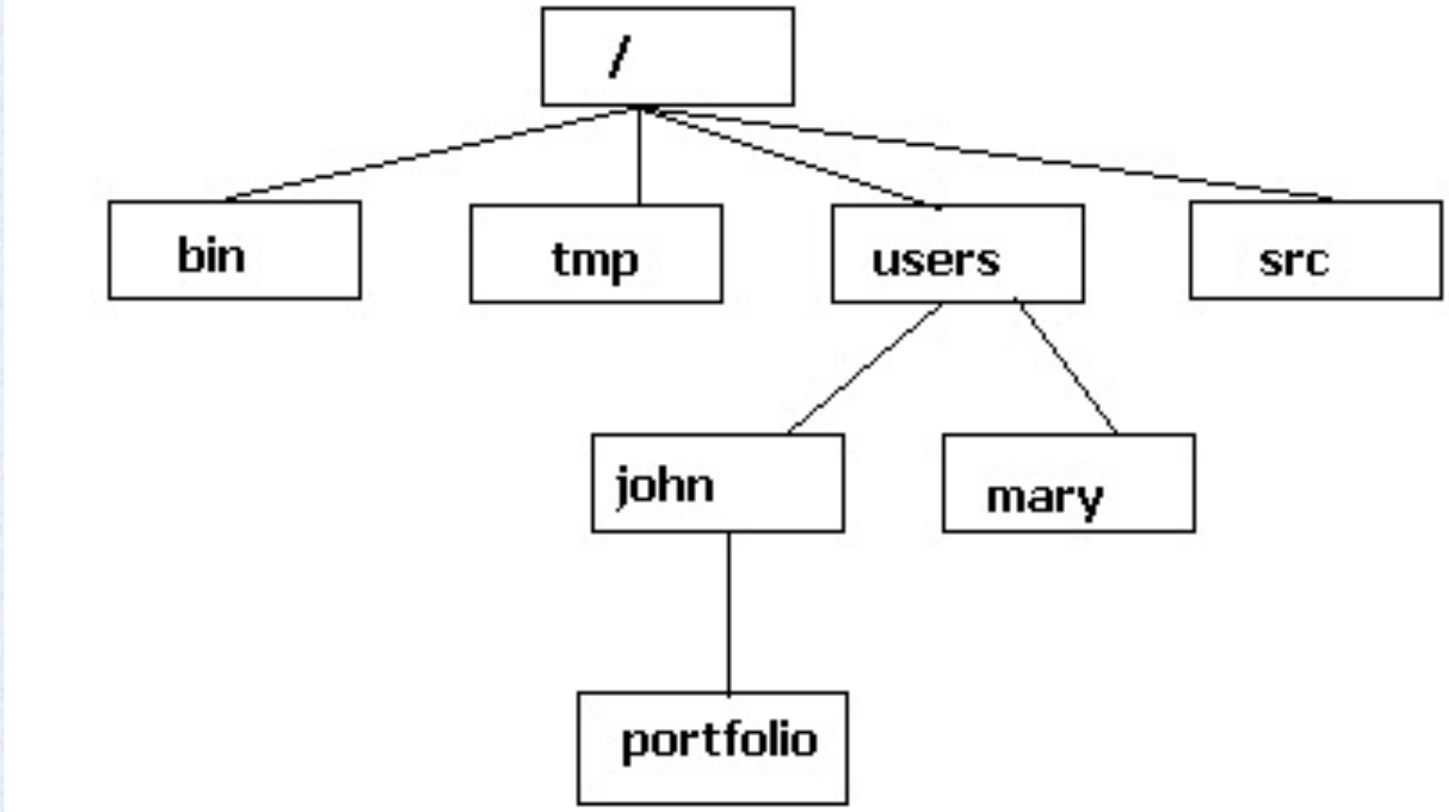


GETTING HOME AGAIN

```
$ pwd
/users/john/
$ cd ..
$ pwd
/users
$ cd ..
$ pwd
/
$ cd
$ pwd
/users/john
$
```

CREATING A DIRECTORY

```
$ pwd
/users/john
$ mkdir portfolio
$ cd portfolio
$ pwd
/users/john/portfolio
$
```



RELATIVE PATHNAMES

```
$ pwd
/users/john/portfolio
$ cd ../../mary
$ pwd
/users/mary
$
```

```
$ pwd
/users
$ cd john/portfolio
$ pwd
/users/john/portfolio
$
```

ABSOLUTE PATHNAMES

```
$ pwd
/users/john
$ cd /users/mary
$ pwd
/users/mary
$ cd /tmp
$ pwd
/tmp
$ cd /users/john/portfolio
$ pwd
/users/john/portfolio
$
```

DIRECTORY COMMANDS

Directories

mkdir *directory-name* make a new directory

rmdir *directory-name* remove a directory (must be empty of all files)

cd *directory-name* change to a directory

cd change to your home directory

cd .. change to directory one level up

~ home directory

. current directory

.. directory one level back

/ entire filesystem root

FILE COMMANDS

Working With Files

ls *options filename* list files in a directory

Options for **ls** include:

- **-a** show all files including hidden ones (those beginning with a ".")
- **-l** long list showing ownership, permission and links
- **-t** time-ordered list
- **-F** mark directories with "/", links with "@" and executables with "*"

mv *options filename new-filename* rename a file

mv *options filename directory-name*
move file to a new directory

mv *options directory-name new-directory-name*
move a directory and all of its contents to a new directory, keeping the structure intact. New directory must already exist.

cp *options filename new-filename* copy file

rm *options filename* delete file

The most useful option for **mv, cp,** and **rm** is **-i.** This will cause command to verify (inquire) before execution.

File Manipulation

Command	Description
vi [f]	*Vi fullscreen editor*
emacs *[f]*	*Emacs fullscreen editor*
ed *[f]*	*Text editor*
wc *f*	*Line, word, & char count*
cat *f*	*List contents of file*
more *f*	*List file contents by screen*
cat *f1 f2 > f3*	*Concatenates f1 & f2 into f3*
chmod *mode f*	*Change protection mode of f*
cmp *f1 f2*	*Compare two files*
cp *f1 f2*	*Copy file f1 into f2*
sort *f*	*Alphabetically sort f*
split *[-n] f*	*Split f into n-line pieces*
mv *f1 f2*	*Rename file f1 as f2*
rm *f*	*Delete (remove) file f*
grep *'ptn' f*	*Outputs lines that match ptn*
diff *f1 f2*	*Lists file differences*
head *f*	*Output beginning of f*
tail *f*	*Output end of f*

MORE FILE COMMANDS

more, not **page** on our system

Viewing File Contents

page *filename* go through file one page at a time
less *filename* page through file with capability to go backwards

head *options filename* view the first few lines
tail *options filename* view the last few lines

Printing

To select a new printer:

LPDEST *printername*
export LPDEST

lp *option filename* print file
cancel *printjob* cancel print job in queue. Get printjob name from **lp** command result
enscript *options filename* format file for PostScript printer
lpstat –p *printername* check printer queue. Use printername "all" to see full list.

bl-soic-1035-1

CHECKING PERMISSIONS

```
$ ls -l apple.txt
-rwxr--r--   1 december december        81 Feb 12 12:45 apple.txt
$
```

The sequence **-rwxr--r--** tells the permissions set for the file apple.txt. The first **-** tells that apple.txt is a file. The next three letters, **rwx**, show that the **owner** has **read, write, and execute** permissions. Then the next three symbols, **r--**, show that the **group** permissions are read only. The final three symbols, **r--**, show that the **world** permissions are read only.

CHANGING PERMISSIONS

Permissions and Security

chmod *permission filename(s)* set level of access.

permissions pertain to (u)ser, (g)roup and (o)ther and include (r)ead, (w)rite, and e(x)ecute.

Turn access on with “+”

Turn access off with “-”

Set access exactly to with “=“

chmod go-rwx *filename(s)* protect your files from prying eyes

CHMOD UNDERSTANDS OCTAL

To lock out others:

```
$ chmod 700 apple.txt
$
```

If someone else tries to look into apple.txt, they get an error message:

```
$ cat apple.txt
cat: apple.txt: Permission denied
$
```

To allow access to others:

```
$ chmod 744 apple.txt
$
```

FILE PERMISSIONS

The chmod command uses as an argument a string which describes the permissions for a file. The permission description can be in the form of a number that is exactly three digits. Each digit of this number is a code for the permissions level of three types of people that might access this file:

1. Owner (you)
2. Group (a group of other users that you set up)
3. World (anyone else browsing around on the file system)

The value of each digit is set according to what rights each of the types of people listed above have to manipulate that file.

Permissions are set according to numbers. Read is 4. Write is 2. Execute is 1. The sums of these numbers give combinations of these permissions:

- 0 = no permissions whatsoever; this person cannot read, write, or execute the file
- 1 = execute only
- 2 = write only
- 3 = write and execute (1+2)
- 4 = read only
- 5 = read and execute (4+1)
- 6 = read and write (4+2)
- 7 = read and write and execute (4+2+1)

ENVIRONMENT CONTROL

Shell Commands

history show previously-run commands

escape (repeat **k** or **j**) move up or down through history of commands. Press "enter" to re-execute (ksh only).

alias *newcommand command-string* create new command that runs *command-string*, including all of the given options

command **>** *outputfile* *command*'s output replaces *outputfile* (**>>** will append)

command **<** *inputfile* *command*'s input comes from *inputfile*

command1 **|** *command2* *command1*'s output becomes *command2*'s input

no **escape** command in our (bash) shell

pipe operator

Environment Control

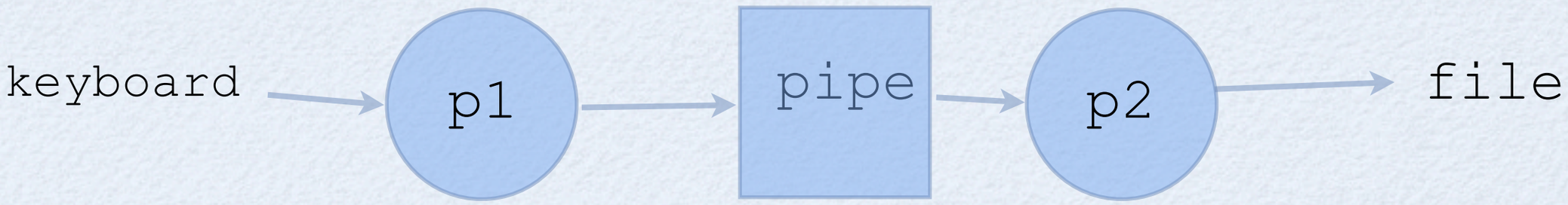
Command	Description
cd *d*	*Change to directory d*
mkdir *d*	*Create new directory d*
rmdir *d*	*Remove directory d*
mv *f1 [f2...] d*	*Move file f to directory d*
mv *d1 d2*	*Rename directory d1 as d2*
passwd	*Change password*
alias *name1 name2*	*Create command alias*
unalias *name1*	*Remove command alias name1*
rlogin *nd*	*Login to remote node*
logout	*End terminal session*
setenv *name v*	*Set env var to value v*
unsetenv *name1 name2...]*	*remove environment variable*

PIPES

- Processes have three **pipes** assigned to them when they are created
  - **stdin** - standard input
    - by default connected to terminal keyboard
  - **stdout** and **stderr -** standard output and standard error
    - by default connected to terminal output
- Input and output redirection operators can connect them to files instead.

PIPE OPERATOR EXAMPLE

```
$ p1 | p2 > file
```



PROCESS CONTROL

Working With Processes

ps *option* show running jobs on system.

ps -fu *userid* show jobs running as *userid*

kill *pid* kill job given by process id *pid*.
Use **ps -fu** *userid* to find the pid.

control-z stop the foreground job

bg (fg) put a stopped job into the background (foreground)

jobs show user's running and suspended jobs

nohup *command* & run *command* in the background; it will not stop when you logout. (applies to ksh only)

Process Control

Command	Description
Ctrl/c *	*Interrupt processes*
Ctrl/s *	*Stop screen scrolling*
Ctrl/q *	*Resume screen output*
sleep *n*	*Sleep for n seconds*
jobs	*Print list of jobs*
kill [%n]	*Kill job n*
ps	*Print process status stats*
kill -9 *n*	*Remove process n*
Ctrl/z *	*Suspend current process*
stop %*n*	*Suspend background job n*
command&	*Run command in background*
bg [%*n*]	*Resume background job n*
fg [%*n*]	*Resume foreground job n*
exit	*Exit from shell*

In **bash** (our lab's default shell), just append & to a command to run it in the background (w/o logout protection). Don't use **nohup**.

PS COMMAND

The `ps -x' command will list all your currently-running jobs. An example is:

```
  PID TT STAT  TIME COMMAND
 6799 co IW    0:01 -csh[rich] (csh)
 6823 co IW    0:00 /bin/sh /usr/bin/X11/startx
 6829 co IW    0:00 xinit /usr/lib/X11/xinit/xinitrc --
 6830 co S     0:12 X :0
 6836 co I     0:01 twm
 6837 co I     0:01 xclock -geometry 50x50-1+1
 6841 p0 I     0:01 -sh[rich on xterm] (csh)
 6840 p1 I     0:01 -sh[rich on xterm] (csh)
 6842 p2 S     0:01 -sh[rich on login] (csh)
 6847 p2 R     0:00 ps -x
```

The meaning of the column titles is as follows:

```
PID         process identification number
TT          controlling terminal of the process
STAT        state of the job
TIME        amount of CPU time the process has acquired so far
COMMAND     name of the command that issued the process
```

```
R=Runnable
T=Stopped
S=Suspended-sleep<20s
I=Idle-sleep>20s
```

http://heather.cs.ucdavis.edu/~matloff/UnixAndC/Unix/Processes.pdf

TAR

Though it is seldom used with tapes any more, old **t**ape **ar**chive utility, **tar**, is still the standard Unix way of creating an archive (in preference to other archive formats such as zip),. Though not shown in the example below, it does nested directories as well as files of any kind.

```
$ tar -cvf lab.tar *.c
save.c
test.c
$ ls
lab.tar   save.c    test.c
$ mkdir lab
$ cd lab
$ tar -xf ../lab.tar
$ ls
save.c  test.c
```

Why use a TUI?

- **T**extual **U**ser **I**nterfaces (TUI) interaction has a number of advantages over using a GUI.
  - often more efficient to use (if you're proficient)
  - simple to program
  - simple to use remotely
  - necessary for operations without a GUI
  - can be incorporated into **scripts** (programs)