The window manager on your computer has been disabled, which means that you will not have a graphical user interface available.

Instead, you will begin to learn how to use the command line interface.

1. Log into your computer. You will get a prompt that is similar to the prompt in a terminal window.
2. The program that is running is called a *shell*, or command line interpreter. To see which shell is running, you can examine the shell variable `$0`.

   ```
   echo $0
   ```

   It should be bash. The hyphen in the name indicates that this is a login shell rather than one that was started after logging in.
3. The shell keeps track of a *current working directory*. File locations are taken relative to this directory, so it it important to know which directory is current. When you first log in, the current directory is your home direction. To display the name, use the `pwd` command (print working directory).

   ```
   pwd
   ```
4. *Directories* (also know as folders) contain files and other directories (called *subdirectories*). To list the files in the current directory, use the `ls` command.

   ```
   ls
   ```
5. To get more information about files, including the size, ownership and permissions, use the `ls` command with the `-l` option.

   ```
   ls -l
   ```
6. Files whose names begin with a period are hidden from view. To see all the files in the current directory, use `ls` with the `-a` option.

   ```
   ls -a
   ```

   Many of these files were created when you first logged in.

   At the beginning of the list, you will see two special names.

   ```
   . ..
   ```

   `.` is an alias for the name of the current directory and `..` is an alias for the name of the parent of the directory (one level up in the directory hierarchy).
7. Try these commands:

   ```
   ls .
   ls ..
   ```

8. Unix commands usually have options which are indicated by a hyphen followed by a single letter (like `ls -a`) or two hyphens followed by a full word (like `ls --all`). In addition, many commands take additional arguments (like `ls -l ..`). Frequently, but not always, additional arguments are file names. If a command is written properly, you can mix and match options and other arguments and they can occur in any order.

   Try these variations:

   ```
   ls -a -l
   ls -al
   ls -a -l ..
   ls .. -al
   ```

9. The command to create a directory is `mkdir`. Create a directory for this course.

   ```
   mkdir cs246
   ```

10. Use ls to list files in the directory.

    ```
    ls -a cs246
    ```

    Although you haven't added any files, it will contain `.` and `..`

11. The `-d` option for `ls` makes it list the directory itself instead of the files inside it.

    ```
    ls -ld cs246
    ```

    This gives a long listing the leftmost character is the type of the file. For an plain (ordinary) file, the type is indicated with a hyphen. For a directory, the type is `d`.

12. To change the current directory, use the command `cd`. Change to your cs246 directory.

    ```
    cd cs246
    ```

13. Use pwd to show the new current directory.

    ```
    pwd
    ```

14. Now let's create a file. The touch command will create an empty file.

    ```
    touch foobar
    ```

15. List the current directory.

    ```
    ls -al
    ```

    You'll see that the file `foobar` has size 0. The size field comes just before the time field.

16. The `mv` command can either rename files or move them. The first argument is an existing file. If the second argument is a file, the first is renamed. If the second argument is a directory, the file is moved into the directory. Rename `foobar` to `xyzzy` and then use `ls` to list your files.

```
mv foobar xyzzy
ls
```

17. Now we'll delete the file using `rm` and then use `ls` to list the current directory (which should be empty).

```
rm xyzzy
ls
```

18. The `seq` command (short for sequence) generates a sequence of integers. It is extremely useful for generating test data for programs. Try it out. With one argument (call it n), it prints the integers from 1 to n.

```
seq 10
```

    With two arguments (call them m and n) it prints the integers from m to n.

```
seq 5 10
```

19. When a program starts, it is connected to an input stream called standard input and an output stream called standard output. These are both connected to your terminal by default. You can redirect standard output to a file like this: *`command > file`*. Think of `>` as an arrow pointing to the file.

    Run `seq` and redirect the output to a file called data1.

```
seq 5 > data1
```

    Create two more files as well.

```
seq 6 10 > data2
seq 11 17 > data3
```

20. To display the contents of the files, we'll use the `cat` command. `cat` reads from files given as arguments and prints the contents of each to standard output.

```
cat data1 data2 data3
```

21. You can also use a wild card to specify the files.

```
cat data*
```

    `cat` is short for concatenate, which means to join items together in a chain.

    `cat` can also be used to display the contents of a single file.

```
cat data1
```

22. If you don't give `cat` any file names, it will read from standard input. If standard input is the terminal, you will need to signal end-of-file when you are done typing. To signal end-of-file, use Ctrl-d at the beginning of a line.

    Use `cat` to create another data file by running it with no arguments, redirecting output.

    ```
    cat > data4
    18
    19
    20
    Ctrl-d
    ```

    Show the contents of the file you just created with `cat`.

    ```
    cat data4
    ```

23. Now we'll concatenate all of the data files.

    ```
    cat data*
    ```

24. Now we'll investigate pipelines. pipelines connect the standard output from one program to the standard input for another. We'll use a pipeline to display the first 10 lines of the concatenation.

    ```
    cat data* | head
    ```

25. And the first 5 lines.

    ```
    cat data* | head -n 5
    ```

26. And all but the last 5 lines.

    ```
    cat data* | head -n -5
    ```

27. Display the last 5 lines.

    ```
    cat data* | tail -n 5
    ```

28. And the lines starting with line 5.

    ```
    cat data* | tail -n +5
    ```

29. Materials for this course are stored in the directory `/home/mathcs/courses/cs246`. These include date files and test programs for various assignments.

    A *pathname* is the full name of a file, including the directory it is in and all directories above. All files are stored below the root directory which is called `/`.

    A *relative* pathname is one the does not begin with a slash. Relative pathnames are taken relative to the current directory.

    An *absolute* pathname is one that does begin with a slash. Absolute paths are relative to the root directory.

    List the files in the course directory.

```
ls /home/mathcs/courses/cs246
```

30. One of the files in the course directory is called computers.txt. It is a book about computers from 1962. Use `cat` to display the file. Do not retype the entire file name. Use the uparrow key to bring back the previous command and edit it.

```
cat /home/mathcs/courses/cs246/computers.txt
```

31. This file was very long. Use the `wc` command to find out how long. Again, do not even think about retyping the entire command.

```
wc /home/mathcs/courses/cs246/computers.txt
```

`wc` (word count) displays the numbers of words, lines, and characters in the file. This one is almost 10000 lines long (it is an entire book).

32. `cat` does not work very well for displaying long files, particularly if you do not have a scroll bar. Instead, use the program `less`

```
less /home/mathcs/courses/cs246/computers.txt
```

`less` displays one screenful of a file at a time. Try out these ways of moving around in the file.

space bar	Move down one page
down arrow	Move down one line
up arrow	Move up one line
`G`	Move to the end of the file
`1G`	Move to the top of the file
n`G`	Move to line n

33. With less you can also search for a particular string. To search for a string, type a (forward) slash followed by the string. Searches are case sensitive (in fact, they use regular expressions, but more on that later).

    Search for the string roadblock using

    ```
    /roadblock
    ```

    Read the paragraph and think about the fact that today we have chips with over a trillion transistors.

34. Now move back to the top of the file using

    ```
    1G
    ```

35. Try moving to the end of the file using

    ```
    $G
    ```

36. Try moving to line 200 using

    ```
    200G
    ```

37. You can leave `less` by typing `q`.

38. Make a subdirectory of your cs246 directory called `data`. Move the files `data1` and `data2` into `data` and check that they are there.

    ```
    mv data1 data2 data
    ls data
    ```

39. Move the remaining data files into `data` like this (make sure that there is not a space between `data` and `*`):

    ```
    mv data* data
    ```

    `data*` matches any file whose name begins with data. This command gives an error message (because it is also trying to move `data` into itself which doesn't make sense). However, it will have moved `data3` and `data4`.

40. Now use `cd` to change your directory to `data` and list the files.

```
cd data
ls
```

41. Use `ls` to list the files in the directory one level up. The *data* directory should be there. Remember that `..` refers to the directory that is one level up.

```
ls ..
```

42. Now we'll move the data files back to your cs246 directory. Then move back up a level and list the files.

```
mv data* ..
cd ..
ls
```

43. Create one more file in the data directory using `touch` and use `ls` to verify that it is there.

```
touch data/foo
ls data
```

44. Next we'll try remove the `data` directory using the `rmdir` command.

```
rmdir data
```

This fails because `rmdir` will only remove a directory if the directory is empty.

45. Use `rm` to remove `foo` from the `data` directory. Then try again to remove `data`. Use `ls` to verify that it is gone.

```
rm data/foo
rmdir data
ls
```

46. Use a pipeline to count the distinct words in the file `/home/mathcs/courses/cs246/computers.txt`. First, cd to the directory.

```
cd /home/mathcs/courses/cs246
```

t. First we'll convert to all uppercase letters to lowercase (and pipe the results through less).

```
tr A-Z a-z < computers.txt | less
```

47. Next we'll convert all the nonalphabetic characters to newlines.

```
tr A-Z a-z < computers.txt | tr -c a-z '\n' | less
```

48. Now we'll sort the results. You'll need to move down in the file a very long way since there are now lots of blank lines, and they come first.

```
tr A-Z a-z < computers.txt | tr -c a-z '\n' | sort | less
```

49. To remove all but one of the blank lines, use

```
tr A-Z a-z < computers.txt | tr -c a-z '\n' | sort | uniq | less
```

50. Let's get rid of the blank line at the top using tail.

```
tr A-Z a-z < computers.txt | tr -c a-z '\n' | sort | uniq |
    tail -n +2 | less
```

51. Finally we can the number of distinct words by counting lines.

```
tr A-Z a-z < computers.txt | tr -c a-z '\n' | sort | uniq |
    tail -n +2 | wc
```

52. Emacs is a very powerful text editor. Among other things, it will indent your programs properly. To run emacs you can use `emacs file`. This will create the file if it does not already exist. You can also start emacs without a file. Let's try that.

```
emacs
```

54. While you are running emacs, do

```
Ctrl-z
```

. This will suspend emacs and return you to the shell. While emacs is suspended, you can run other commands. Use the command

```
jobs
```

to get a list of suspended jobs. To return to emacs, use the command

```
fg
```

This means foreground. Jobs that do not need terminal input or output can also run in the background, independently of your terminal.

55. When you get back to emacs, start the emacs tutorial with

```
Ctrl-h t
```

Emacs will tell you how to move around, save files, and exit. Work through the entire tutorial.