Useful Linux Commands and Text Editors

CISC 3350 — CUNY Brooklyn College
Lecture Notes

- . ./.long_prompt
- pwd
- whoami
- ls
- ls -a
- ls -l
- ls -al
- ls dir
- cd
- cd .
- cd ~
- cd /
- cd dir
- mkdir
- touch
- rmdir
- rm
- cat
- cp
- mv
- sudo
- su
- sudo -i
- man
- info
- ping
- wget
- gcc
- ./executableFile
- gdb
- exit
- vim
- emacs
- nano

Top

Useful Linux Commands

The command:

```
$ pwd
```

stands for 'print working directory' and shows the current folder in which the terminal is positioned. An example of output from this command is `/home/cisc3350/Desktop`, which means that the terminal is currently located in the Desktop folder.

Output example:

```
$ pwd
/home/cisc3350/Desktop
```

Useful Linux Commands

The command:

```
$ whoami
```

outputs the username of the user that is currently logged into the device/account and are using this terminal. Examples of outputs from this command are `miriam`, `student`, `admin`, or `root`.

Output example:

```
$ whoami
Miriam.Briskman12
```

Useful Linux Commands

The command:

```
$ ls
```

without any additional command-line arguments, stands for 'list' (verb) and shows the contents of the current directory. This includes only the names of the various non-hidden files and folders included in the current folder.

Output example:

```
$ ls
cisc_3350 resume.doc text.txt
```

The command:

```
$ ls -a
```

(note that the `-a` is a command-line argument) shows the names of the various files and folders of the current directory, including all the hidden files. A hidden file (or folder) on Linux is one whose name starts with a dot symbol, and examples of hidden files are: `.` (the link to the current directory [itself!],) `..` (the link to the parent directory of the current directory), `.bashrc`, and the `.git` folder that might be found in your directory.

Output example:

```
$ ls -a
. .. .git cisc_3350 resume.doc text.txt
```

Useful Linux Commands

The command:

```
$ ls -l
```

(note that the `-l` like 'Larry' is a command-line argument) shows a long printout of the various files and folders of the current directory. In this printout, the information of each file or folder is displayed on a separate line, and includes metadata such as: permissions on using the file, the number of files inside the folder (in case it is a folder,) the username of the user who owns the file, the group of the user who owns the file, the file size in bytes, the recent modification date of the file, and, of course, the name of the file. Output example:

```
$ ls -l
drwxrwxr-x 76 Miriam.Briskman12 Miriam.Briskman12  4096 Jan 03 15:34 cisc_3350
-rw-rw-r--  1 Miriam.Briskman12 Miriam.Briskman12 34982 Feb 13 09:44 resume.doc
-rw-rw-r--  1 Miriam.Briskman12 Miriam.Briskman12    25 Jan 22 04:44 text.txt
```

Useful Linux Commands

The command:

```
$ ls -al
```

(or `ls -la` or `ls -a -l` or `ls -l -a`) is a combination of `ls -a` and `ls -l`: it shows a long printout of the various files and folders of the current directory, including of all the hidden files. Output example:

```
$ ls -al
drwxrwxr-x   2 Miriam.Briskman12 Miriam.Briskman12  4096 Feb 14 12:24 .
drwxrwxr-x  16 Miriam.Briskman12 Miriam.Briskman12  4096 Jan 22 04:41 ..
drwxrwxr-x 832 Miriam.Briskman12 Miriam.Briskman12  4096 Jan 22 04:41 .git
drwxrwxr-x  76 Miriam.Briskman12 Miriam.Briskman12  4096 Jan 03 15:34 cisc_3350
-rw-rw-r--   1 Miriam.Briskman12 Miriam.Briskman12 34982 Feb 13 09:44 resume.doc
-rw-rw-r--   1 Miriam.Briskman12 Miriam.Briskman12    25 Jan 22 04:44 text.txt
```

Useful Linux Commands

The command:

```
$ ls existingDirectory
```

lists the contents of a directory whose name (or path to) is `existingDirectory`. You can also add the `-a` or `-l` command-line arguments if wanted before or after `existingDirectory`. Here, `existingDirectory` could be either a relative or an absolute path to a directory. An absolute path is one that starts with the root directory, like `/home/cisc3350/Desktop/`, and a relative path is one without the forward slash to its left, like `Desktop/games/`.

Output example:

```
$ ls /home/cisc3350/
helloworld.c errno.c
```

Useful Linux Commands

The command:

```
$ cd
```

(without any additional command-line arguments) stands for 'change directory'. When typed into the terminal, it will change the current directory of the terminal to your user's home directory. The Linux computer saves the home directories of all the users within the file `/etc/passwd`, which you can view at any time. To change the home directory to a different one, you should contact the system administrator of your Linux device/account to change it, or, if the machine belongs to you, you can use the suggested commands at this StackOverflow page to change your home directory to a different one.

An output example is shown on the next slide.

Useful Linux Commands

Output example of `cd`:

```
$ pwd
/home/cisc3350/Desktop/
$ cd
$ pwd
/home/
```

As the above example shows, when `cd` is executed, it usually doesn't print any output (unless an error occurred,) but we can confirm that we landed on the right directory by calling `pwd`.

Useful Linux Commands

The command:

```
$ cd ..
```

(note that `..` here is a command-line argument) will change the current directory of the terminal to the parent directory of the current folder. For example, if `pwd` says that the current folder is `/home/cisc3350/Desktop/`, typing `cd ..` will change the current directory to `/home/cisc3350/`. Note that the root directory, `/` is special. Theoretically, it doesn't have a parent directory, but if you try doing `cd ..` when inside the root, you will return to ... the root directory itself! So it is like a self-loop. Output example:

```
$ pwd
/home/cisc3350/Desktop/
$ cd ..
$ pwd
/home/cisc3350/
```

Useful Linux Commands

The command:

```
$ cd .
```

will change the current directory of the terminal to the current directory itself! That is, you won't leave the current directory, and `cd .` serves as a self-loop.

Output example:

```
$ pwd
/home/cisc3350/
$ cd .
$ pwd
/home/cisc3350/
```

Useful Linux Commands

The command:

```
$ cd ~
```

just as `cd` does, will change the current directory of the terminal to your user's home directory.

Output example:

```
$ pwd
/home/cisc3350/Desktop/
$ cd
$ pwd
/home/
```

Useful Linux Commands

The command:

```
$ cd /
```

will change the current directory of the terminal to the computer's root directory, which is the grandparent of all directories in Linux. Note that we might mention the term 'root' in several contexts: not only do we have a directory called 'root', but we also most likely have a user whose name is `root`, which is the ultimately-privileged user of the computer who can perform almost any action, including installing new applications, deleting the disk, and accessing files of other users. Output example:

```
$ cd /
$ pwd
/
```

Useful Linux Commands

The command:

```
$ cd existingDirectory
```

will change the current directory of the terminal to the directory indicated by `existingDirectory`. Here, `existingDirectory` could be either a relative or an absolute path to a directory. An absolute path is one that starts with the root directory, like `/home/cisc3350/Desktop/`, and a relative path is one without the forward slash to its left, like `Desktop/games/`. Output example:

```
$ pwd
/home/
$ cd cisc3350/Desktop/
$ pwd
/home/cisc3350/Desktop/
```

Useful Linux Commands

The command:

```
$ mkdir newDirectory
```

stands for 'make directory' and creates a new directory whose name is `newDirectory` inside the current directory. For example, if the terminal is currently situated inside `/home/cisc3350/Desktop/`, and you type `mkdir games`, a new folder called `games` will be created inside the `Desktop/` directory (in case it is not present there already; if so, an error message that tells that the directory already exists will be displayed to the terminal.) Output example:

```
$ ls -l
-rw-rw-r-- 1 Miriam.Briskman12 Miriam.Briskman12  25 Jan 22 04:44 text.txt
$ mkdir games
$ ls -l
drwxrwxr-x 2 Miriam.Briskman12 Miriam.Briskman12  4096 Feb 14 13:34 games
-rw-rw-r-- 1 Miriam.Briskman12 Miriam.Briskman12    25 Jan 22 04:44 text.txt
```

Useful Linux Commands

The command:

```
$ touch newFile
```

does the following: if the file named `newFile` already exists in the current directory, its recently modified date will be updated to now. Otherwise, if a file with such a name does not exist, the `touch` will create an empty file called `newFile` in the current directory. For example, typing `touch notes.txt` will create an empty text file called `notes` inside the current directory (in case this file doesn't already exist there.) Output example:

```
$ ls -l
-rw-rw-r-- 1 Miriam.Briskman12 Miriam.Briskman12  25 Jan 22 04:44 text.txt
$ touch notes.txt
$ ls -l
-rw-rw-r-- 1 Miriam.Briskman12 Miriam.Briskman12  0 Feb 14 13:40 notes.txt
-rw-rw-r-- 1 Miriam.Briskman12 Miriam.Briskman12 25 Jan 22 04:44 text.txt
```

Useful Linux Commands

The command:

```
$ rmdir existingDirectory
```

stands for 'remove directory' and will delete the folder named `ExistingFolder` located inside the current directory. If the `existingDirectory` directory contains any files, or if the directory doesn't exist, an error message will be printed to the terminal. In case you wish to erase a non-empty directory, use the command `rm -r existingDirectory` (`rm` = 'remove'; `-r` = 'recursively') to delete this directory and all the files and folders inside it. Output example:

```
$ ls -l
drwxrwxr-x 2 Miriam.Briskman12 Miriam.Briskman12  4096 Feb 14 13:34 games
-rw-rw-r-- 1 Miriam.Briskman12 Miriam.Briskman12    25 Jan 22 04:44 text.txt
$ rmdir games
$ ls -l
-rw-rw-r-- 1 Miriam.Briskman12 Miriam.Briskman12 25 Jan 22 04:44 text.txt
```

Useful Linux Commands

The command:

```
$ rm existingFile
```

stands for 'remove' and will delete the file named `existingFile` located inside the current directory. If the `existingFile` doesn't exist, an error message will be printed to the terminal. An example would be `rm myfile.txt`, using which we can delete a file called `myfile.txt` inside the current directory. Note that, after a file is deleted, it is permanently erased from the computer, unlike files that are deleted on a Windows or a Mac. A GUI version of Linux might feature a 'garbage bin' that temporarily stores deleted files, but the command-line version doesn't have such a feature by default.

An output example is shown on the next slide.

Useful Linux Commands

Output example of `rm`:

```
$ ls -l
-rw-rw-r-- 1 Miriam.Briskman12 Miriam.Briskman12  739 Feb 14 13:40 notes.txt
-rw-rw-r-- 1 Miriam.Briskman12 Miriam.Briskman12   25 Jan 22 04:44 text.txt
$ rm notes.txt
$ ls -l
-rw-rw-r-- 1 Miriam.Briskman12 Miriam.Briskman12 25 Jan 22 04:44 text.txt
```

As we learned a couple slides ago, `rm` can also be used to delete folders.

Useful Linux Commands

The command:

```
$ cat file1 file2 file3...
```

stands for 'concatenate' and will output the contents of all the files listed after `cat` to the screen. You can't edit the contents of these files using `cat`, but you can see each file's full content, regardless of how large the file is. Once the content of the files is output, the command prompt (e.g., `$`, which is the line in Linux that awaits your commands) will return back to you to get further commands from you. An example of using `cat` with a single file is by typing `cat myfile.txt`, which will show the contents of `myfile.txt` on the screen.

An output example is shown on the next slide.

Useful Linux Commands

Output examples of `cat`:

```
$ cat myfile.txt
Hello, World!
$ cat message.data myfile.txt helloworldprog.c
Today is a wonderful day!
Hello, World!
#include <stdio.h>
int main()
{
    printf ("Hello, World!\n");
    return 0;
}
```

Useful Linux Commands

The command:

```
$ clear
```

brings the command prompt (e.g., $, which is the line in Linux that awaits your commands) to the top of the terminal in a way that is similar to 'clearing' the commands and data that were previously output to the screen. After typing this command, there is going to be nothing on the screen besides the command prompt.

Output example: Calling

```
$ clear
```

will result in the screen looking as follows, waiting for you to enter new commands:

```
$
```

Useful Linux Commands

The command:

```
$ cp path/To/ExistingFile path/To/The/Newly/CreatedCopyFile
```

stands for 'copy' and creates a copy of the file that is found in the directory `path/To/` called `ExistingFile` in the destination directory `path/To/The/Newly/` under the name `CreatedCopyFile`. That is, the original file `ExistingFile` remains unchanged, and we just create a 'clone' of this file inside a different folder (and, possibly, with a new name.)

Output example:

```
$ ls
cisc_3350 resume.doc text.txt
$ cp ../../photos/vacation.jpg ./vacation-copy.jpg
$ ls
cisc_3350 resume.doc text.txt vacation-copy.jpg
```

Useful Linux Commands

The command:

```
$ mv path/To/ExistingFile path/To/The/Newly/CreatedCopyFile
```

stands for 'move' and performs two actions: (1) it creates a copy of the file `ExistingFile` that is found in the directory `path/To/` and pastes it in the directory `path/To/The/Newly/` under the name `CreatedCopyFile` (2) afterwards, it deletes the original file `ExistingFile` from the `path/To/` directory. This is essentially the action of 'moving' a file.

Output example:

```
$ mv ../../photos/party.jpg .
$ ls
cisc_3350 party.jpg resume.doc text.txt vacation-copy.jpg
$ ls ../../photos/
vacation.jpg graduation.jpg
```

Useful Linux Commands

The command:

```
$ sudo someCommand
```

stands for 'super user do' and lets you execute the command `someCommand` with elevated privileges. The reason that the execution of some commands, such as those that install a new program, requires the call to `sudo` is that these actions are considered sensitive and might bring some danger to the computer if done incorrectly. As such, not all users are permitted to perform such actions. One way to perform such an action is by calling `sudo`, which temporarily elevates the user's privileges to those of the 'super user'.

However, not all users can use the `sudo` command: for some of you, typing `sudo` will result in an error message like: "user x is not on the sudoers list", which means that you can't request to elevate your privileges to perform that action. In such a case, if the action is necessary, the user should contact the system administrator and request that they add the user to the sudoers list, if the organization/institution allows. An example of using `sudo` is typing `sudo apt-get update`, which updates all the installed software on the Linux device to the latest versions.

Useful Linux Commands

Important Note about sudo: If you are a regular/unprivileged Linux user (for example, if you are using your Brooklyn College Linux Account,) you might have no permission to use the `sudo` command, as explained in the paragraph above. No homework assignment or exam will ask you to use the `sudo` command, so don't worry about this current inability to use `sudo`. If you have access to a Linux virtual machine in which you are an administrator user, you should be able to use `sudo`.

The command:

```
$ su
```

stands for 'super user' and elevates your user privileges until you exit the super user mode with the Linux `exit` command. If, when typing this command, you receive an error message like:
`su: Authentication failure`
but you know that your user is on the sudoers list, you might instead type the command `sudo -i` below. If you aren't on the sudoers list, you can't use `su`.

Useful Linux Commands

The command:

```
$ sudo -i
```

is an alternative to typing `su` and elevates your user privileges until you exit the super user mode with the Linux `exit` command. In many cases, the computer's super user is the user whose name is `root`, so you actually switch to the `root`'s account by entering `sudo -i`. One quick way to recognize that you are in 'super user mode' is to note that the symbol at the end of the command prompt is the number sign, `#`, and not the dollar sign, `$`.

If you aren't on the sudoers list of the Linux computer, you can use neither `su` nor `sudo -i`. The following output example shows that you can't use `sudo -i`:

```
$ sudo -i
[sudo] password for Miriam.Briskman12:
Miriam.Briskman12 is not in the sudoers file.
```

Useful Linux Commands

If you are on the sudoers list of the Linux computer, you'll see the following output when you call `sudo -i`:

```
$ sudo -i
[sudo] password for Miriam.Briskman12:
#
```

Seeing the # prompt symbol instead of the $ symbol means that you are now granted the permissions of the privileged `root` user and can call various sensitive commands, such as installing new programs, connecting or disconnecting devices, configuring the Linux computer, and accessing data and files of other users.

Useful Linux Commands

The command:

```
$ man someCommand
```

stands for 'manual' and opens the manual page for the Linux command `someCommand`. For example, entering `man cp` will show the manual page for the 'copy' command. A manual page includes the description of the command, the list of arguments that the command accepts, and any other essential details for the use of the command. If the manual page is longer than the screen, you can scroll down by pressing and holding the down arrow of your keyboard. To exit the manual page and return to the terminal, press the key 'Q' (stands for 'quit') at any time.

An example of the manual page for `cp` is shown on the next slide. This page can be accessed by typing:

```
$ man cp
```

Useful Linux Commands


```
CP(1)                                User Commands                                CP(1)

NAME
       cp - copy files and directories

SYNOPSIS
       cp [OPTION]... [-T] SOURCE DEST
       cp [OPTION]... SOURCE... DIRECTORY
       cp [OPTION]... -t DIRECTORY SOURCE...

DESCRIPTION
       Copy SOURCE to DEST, or multiple SOURCE(s) to DIRECTORY.

 Manual page cp(1) line 1 (press h for help or q to quit)
```

Useful Linux Commands

The command:

```
$ info someCommand
```

opens an information page for the Linux command `someCommand`, which usually contains more information, usage recommendations, and examples about the command than the manual page for this command does. For example, entering `info cp` will show the information page for the 'copy' command. You can navigate an information page in the same way as you do a manual page, by using the arrow keys (to move up and down the page) and the 'Q' key to quit the page.

An example of the info page for `cp` is shown on the next slide. This page can be accessed by typing:

```
$ info cp
```

Useful Linux Commands


```
File: coreutils.info,  Node: cp invocation,  Next: dd invocation,  Up: Basic operations

11.1 'cp': Copy files and directories
=====================================

'cp' copies files (or, optionally, directories).  The copy is completely
independent of the original.  You can either copy one file to another,
or copy arbitrarily many files to a destination directory.  Synopses:

     cp [OPTION]... [-T] SOURCE DEST
     cp [OPTION]... SOURCE... DIRECTORY
     cp [OPTION]... -t DIRECTORY SOURCE...

Welcome to Info version 5.1. Type h for help, m for menu item.
```

Useful Linux Commands

The command:

```
$ ping serverName
```

stands for 'Packet Internet Groper' and checks if the server (essentially, a remote device or webpage) whose name, link, or IP address is `serverName` is responsive over a network, like the Internet. For instance, typing `ping www.google.com` will initiate a connection with the Google website; if the connection is successful, your Linux device and Google will start exchanging data. Otherwise, the data that your Linux device sends won't be 'answered' by Google. This exchange of data will continue endlessly, so you need to press Ctrl + C to finish the exchange and return the command prompt back to you. After you press Ctrl + C, the overall data exchange statistics will be displayed on the terminal before the prompt is returned to you.

An example of pinging `www.google.com` is shown on the next slide.

Useful Linux Commands


```
$ ping www.google.com
PING www.google.com (142.251.40.132) 56(84) bytes of data.
64 bytes from lga25s80-in-f4.1e100.net (142.251.40.132): icmp_seq=1 ttl=111 time=4.45 ms
64 bytes from lga25s80-in-f4.1e100.net (142.251.40.132): icmp_seq=2 ttl=111 time=4.37 ms
64 bytes from lga25s80-in-f4.1e100.net (142.251.40.132): icmp_seq=3 ttl=111 time=4.46 ms
64 bytes from lga25s80-in-f4.1e100.net (142.251.40.132): icmp_seq=4 ttl=111 time=4.48 ms
64 bytes from lga25s80-in-f4.1e100.net (142.251.40.132): icmp_seq=5 ttl=111 time=4.40 ms
^C
--- www.google.com ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4007ms
rtt min/avg/max/mdev = 4.378/4.437/4.486/0.040 ms
$
```

Useful Linux Commands

The command:

```
$ wget someRemoteFile
```

stands for 'web get' and copies over a file whose URL is `someRemoteFile` from a remote server (e.g., a website). This file could be anything, including text files, Word files, C source files, and even HTML pages (webpages). After this command runs, if the output says that "file such and such saved", it means that the file was successfully copies to the current directory where the terminal is located. Otherwise, either the file doesn't exist, a bad URL was typed, or there is no Internet connection. Another possibility is that the `wget` program is not installed on your device, in which case you could install it on your device (if you know you have permissions to use the `sudo` command) by typing `sudo apt-get update` followed by `sudo apt-get install wget` and then following the installation commands on the screen.

An example of using `wget` is shown on the next slide.

Useful Linux Commands


```
$ wget https://www.sci.brooklyn.cuny.edu/~briskman/cisc/3350/prog1.c
--2024-02-20 02:22:17--  https://www.sci.brooklyn.cuny.edu/~briskman/cisc/3350/prog1.c
Resolving www.sci.brooklyn.cuny.edu (www.sci.brooklyn.cuny.edu)... 127.0.0.1
Connecting to www.sci.brooklyn.cuny.edu (www.sci.brooklyn.cuny.edu)|127.0.0.1|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 771 [text/plain]
Saving to: ‘prog1.c’

100%[==================================================>] 771         --.-K/s   in 0s

2024-02-20 02:22:17 (16.7 MB/s) - ‘prog1.c’ saved [771/771]

$ ls
prog1.c
```

Useful Linux Commands

The command:

```
$ grep "pattern" existingDirectory
```

stands for 'global regular expression print' and outputs the names of the files where `pattern`, which is either a simple search string or some regular expression, is present within the `existingDirectory` folder. For example, entering `grep "Bla Bla!" *` will show all the files in the current directory where the string `Bla Bla!` is present. This is because the * symbol is a 'wild-card' that selects all the files in the current directory for this search. The `grep` command has many useful command-line options, which you can learn more about in the `man grep` or `info grep` manual pages.

Many examples of using `grep` are brought at https://www.thegeekstuff.com/2009/03/15-practical-unix-grep-command-examples/.

Useful Linux Commands

The command:

```
$ gcc -Wall -Wextra -O2 -g -o program program.c
```

compiles the C source code located inside the file `program.c` and creates an executable (= binary) file called `program`. The command `gcc` invokes the GNU C Compiler. Above, if we were to omit "`-o program`" from the command ("o" is the lowercase "Ohh" letter,) the executable's name would be the generic name "`a.out`". The command options "`-Wall`" and "`-Wextra`" let the compiler display all the basic and extra possible compilation warnings to alert the user that a certain part of the program might not execute as intended in case of a warning showing up after running the command. The option "`-O2`" ("O" is the uppercase "Ohh" letter) sets the optimization level to 2 (= significant but sane) out of 5 possible optimization levels. Finally, the option "`-g`" allows the user to run the C debugger program, `gdb`, after the compilation in case debugging or program tracing is needed.

An example of compiling a C program with `gcc` is shown on the next slide.

```
$ ls
prog1.c
$ gcc -Wall -Wextra -O2 -g -o prog1 prog1.c
$ ls
prog1.c prog1
```

Since no errors or warnings were shown, and since the executable file "`prog1`" was created, we understand that the compiler didn't find any errors in our code.

The command:

```
$ ./executableFile
```

executes (runs) a program whose name is `executableFile` and that is located in the current directory using the terminal. The output that this program produces will be displayed in the terminal.

Useful Linux Commands

In our class, we will be creating and running C programs after we compile them. For example, entering `./prog1` will run the program/application called `prog1` that is located within the current directory, and entering `./a.out` will run the program/application called `a.out` that is located within the current directory. You may, in a similar way, run programs that are located in other directories on your device/account by replacing the dot `.` with the path to the folder where that program that you are trying to run resides. For example, if a program called `my_app` is located in the folder `/home/cisc3350/Desktop`, you can execute it by typing: `/home/cisc3350/Desktop/my_app` in the terminal and then pressing ENTER/RETURN.

An example of running the `prog1` program:

```
$ ls
prog1.c prog1
$ ./prog1
Finished summoning the daemon!
Let it quietly die now.
$
```

Useful Linux Commands

The command:

```
$ gdb ./executableFile
```

runs the GNU C Debugger software whose purpose is to run the program located in the current directory whose name is `executableFile` and trace/debug this program. Once you enter this command, you will enter the debugger's menu. To start running the program, type `run`. If an error occurs during the execution, the debugger will pause the execution and show what the error is and where (inside which function and on which line) it happened. You can then type the `backtrace` (or `bt`, in short) command to show the sequence of function calls starting from the `main` function that describes how this faulty statement was reached. To exit the debugger app, press 'Q'. The manual page of `gdb` features the full list of commands that you can use within the debugger, such as printing the values stored in a variable, creating breakpoints, and other useful commands that you can use to find bugs faster!

A quick introduction to using `gdb` can be found at https://stackoverflow.com/a/2069444/14167156.

Useful Linux Commands

The command:

```
$ exit
```

closes the terminal window. This is an alternative to clicking the 'X' button at the top corner of the terminal application window. Alternatively, if you are remotely connected to a Linux computer, the `exit` command closes the connection and returns you to the command prompt or terminal window of your host device (e.g., Windows, Mac, etc.)

Output example:

```
$ exit
logout
Connection to www.sci.brooklyn.cuny.edu closed.

C:\Users\Miriam Briskman\Desktop>
```

Useful Linux Text Editors

The command:

```
$ vim someFile
```

either creates an empty file named `someFile` within the current directory (if a file with this name doesn't exist) using the `vim` command-line text editor, or opens an existing file named `someFile`. Within the `vim` editor, you can change, delete, and add content.

Here are links to some great `vim` guides and YouTube videos:

- https://www.openvim.com/ -- Interactive vim tutorial
- http://www.vimgenius.com/ -- Another interactive vim tutorial
- https://www.youtube.com/watch?v=ggSyF1SVFr4 -- YouTube: 8-minute Vim Crash Course
- https://www.tutorialspoint.com/vim/index.htm -- Tutorialspoint
- https://www.cs.cmu.edu/~15131/f17/topics/vim/vim-cheatsheet.pdf -- a 2-page vim cheat sheet

Useful Linux Text Editors

The command:

```
$ emacs someFile
```

either creates an empty file named `someFile` within the current directory (if a file with this name doesn't exist) using the `emacs` command-line text editor, or opens an existing file named `someFile`. Within the `emacs` editor, you can change, delete, and add content.

Here are links to some great `emacs` guides and YouTube videos:

- http://www.jesshamrick.com/2012/09/10/absolute-beginners-guide-to-emacs/ -- Absolute Beginner's Guide
- http://ocean.stanford.edu/research/quick_emacs.html -- Quick tutorial
- https://www.youtube.com/watch?v=jPklaqSh3cA -- YouTube: The Basics of emacs (16 mins)
- https://www.youtube.com/watch?v=JdF1QZQqiTI -- YouTube: Emacs GUI tutorial (8 mins)
- https://www.gnu.org/software/emacs/refcards/pdf/refcard.pdf -- a 2-page emacs cheat sheet

Useful Linux Text Editors

The command:

```
$ nano someFile
```

either creates an empty file named `someFile` within the current directory (if a file with this name doesn't exist) using the `nano` command-line text editor, or opens an existing file named `someFile`. Within the `nano` editor, you can change, delete, and add content.

Here are links to some great `nano` guides and YouTube videos:

- https://www.howtogeek.com/42980/the-beginners-guide-to-nano-the-linux-command-line-text-editor/ -- Beginner's Guide
- https://www.tutorialspoint.com/how-to-use-nano-text-editor -- Tutorialspoint
- https://www.youtube.com/watch?v=Jf0ZJZJ8jlI -- YouTube: Nano Text Editor Basics (8 mins)
- https://www.youtube.com/watch?v=gyKiDczLIZ4 -- YouTube: Nano Editor Fundamental (14 mins)
- https://www.nano-editor.org/dist/latest/cheatsheet.html -- Nano cheat sheet

Useful Linux Text Editors

Any questions?

Useful Linux Commands and Text Editors: Sources

Topic 2: Useful Linux Commands and Text Editors. Briskman, Miriam. *CISC 3350 OER Course Packet.* 2024, CC BY-NC 4.0.
URL: https://www.sci.brooklyn.cuny.edu/~briskman/cisc/3350/CISC_3350_OER_Course_Packet.pdf#topic+2:+useful+linux+commands+and+text+editors.

Partly based on the teaching of Professor **Yongqing Xiang** of Brooklyn College.