Traditional Access Permissions

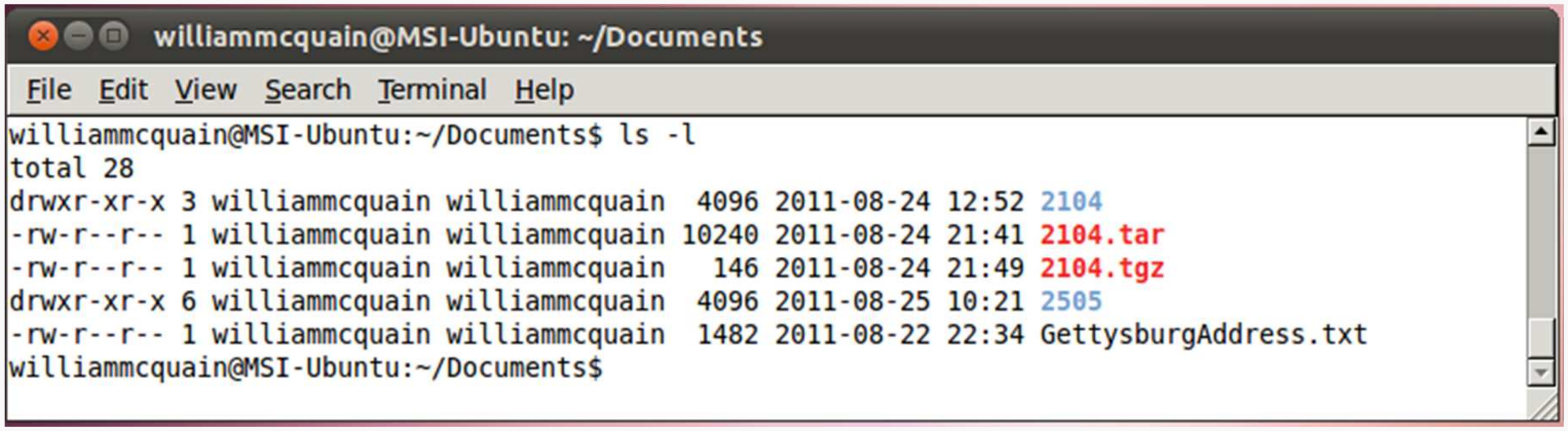
There are three types of users:

- *owner*
- *group*
- *other (aka world)*

A user may attempt to access an ordinary file in three ways:

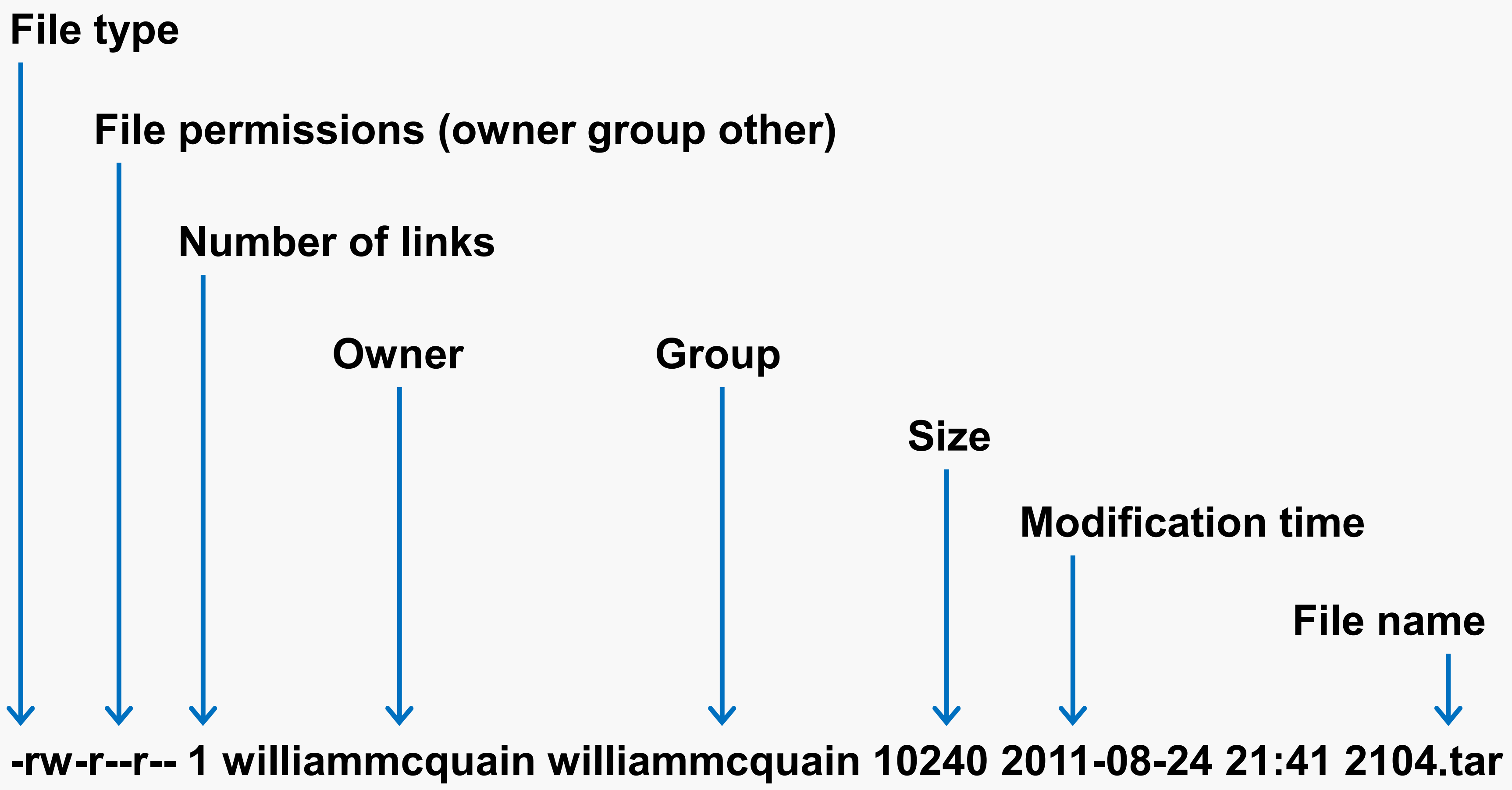
- *read from*
- *write to*
- *execute*

Use **ls –l** to view the file permissions:


williammcquain@MSI-Ubuntu: ~/Documents
File Edit View Search Terminal Help

```
williammcquain@MSI-Ubuntu:~/Documents$ ls -l
total 28
drwxr-xr-x 3 williammcquain williammcquain  4096 2011-08-24 12:52 2104
-rw-r--r-- 1 williammcquain williammcquain 10240 2011-08-24 21:41 2104.tar
-rw-r--r-- 1 williammcquain williammcquain   146 2011-08-24 21:49 2104.tgz
drwxr-xr-x 6 williammcquain williammcquain  4096 2011-08-25 10:21 2505
-rw-r--r-- 1 williammcquain williammcquain  1482 2011-08-22 22:34 GettysburgAddress.txt
williammcquain@MSI-Ubuntu:~/Documents$
```

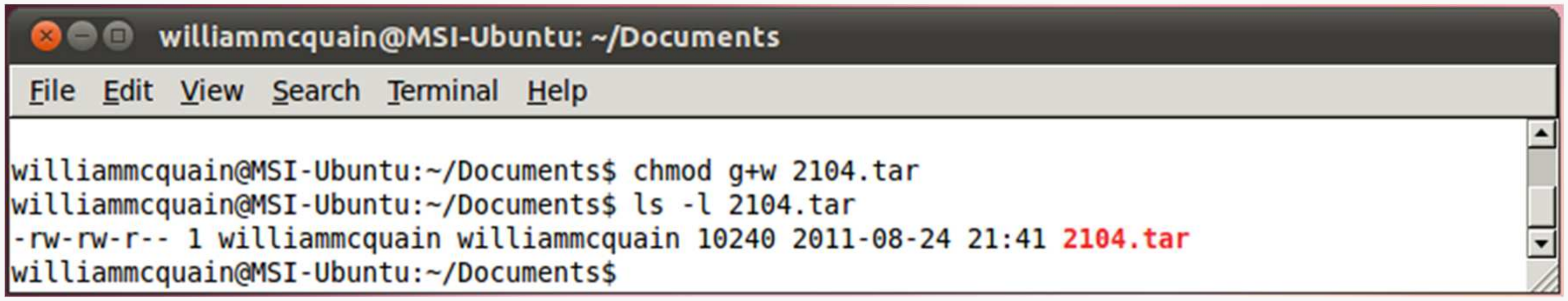
Traditional Access Permissions



```
-rw-r--r-- 1 williammcquain williammcquain 10240 2011-08-24 21:41 2104.tar
```

Changing Access Permissions: chmod

Use the **chmod** command to set or alter traditional file permissions:


williammcquain@MSI-Ubuntu: ~/Documents
File Edit View Search Terminal Help

```
williammcquain@MSI-Ubuntu:~/Documents$ chmod g+w 2104.tar
williammcquain@MSI-Ubuntu:~/Documents$ ls -l 2104.tar
-rw-rw-r-- 1 williammcquain williammcquain 10240 2011-08-24 21:41 2104.tar
williammcquain@MSI-Ubuntu:~/Documents$
```

chmod also allows the use of numeric arguments:

0	no access permissions
1	execute permissions
2	write to permissions
4	read from permissions

So, **chmod 740** would set

owner permissions to **r w x**

group permissions to **r- -**

other permissions to **- - -**

WHY?

Changing Access Permissions: chmod

Binary representations:

none	**0**	**000**
x	**1**	**001**
w	**2**	**010**
r	**4**	**100**

Now notice that **7 = 111** which is the logical OR of **001** and **010** and **100**

And, **740** thus specifies permissions **7** for the owner, **4** for the group and **0** for others.

The Importance of Access Permissions

When working on a shared environment, like the rlogin cluster, it is vital that you make sure that your access permissions are set correctly.

As a general rule, you will rely on the default access permissions, which are controlled via shell configuration files we will discuss later.

When in doubt, use **ls –l** to check!

Removing a File: rm and shred

If you have sufficient permissions, a file can be deleted from the file system by using the **rm** command.

Be very careful with **rm**!

You can also securely remove a file by using the **shred** command, but see Sobell for a discussion of the limitations.

See the discussion of **dd** in Sobell for an alternative way to wipe a file.

Special Characters

Many Linux commands support the use of special characters (aka wildcards) to specify a pattern that identifies a set of files:

?	matches any single character (in the name of an existing file)
*****	matches zero or more characters (in the name of an existing file)
[ ]	matches any of the characters within the braces (in the name of an existing file)

***.txt**

matches any file with extension "txt"

foo?.*

matches a file with any extension and name consisting of "foo" followed by a single character

[abc]foo.html

matches a file with extension "html" and name "afoo" or "bfoo" or "cfoo"

Copying a File Remotely: scp

scp can be used to copy a file between the local machine and a remote machine (or between two remote machines).

For example, the following command would copy `GettysburgAddress.txt` from my computer to a directory named **documents** on **rlogin**:

```
scp  GettysburgAddress.txt   wmcquain@rlogin.cs.vt.edu:documents
```

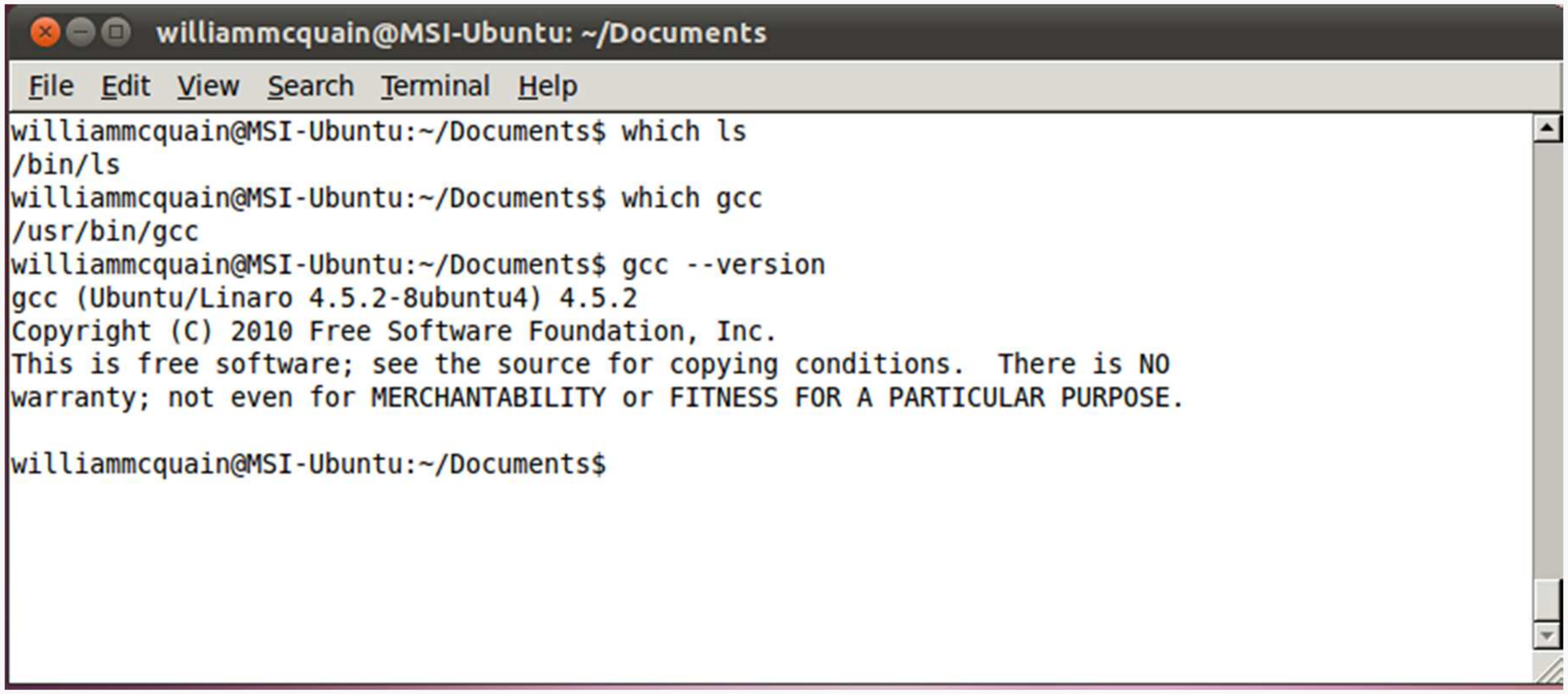
If you haven't set up password-less login, you'll be prompted for the necessary authentication information.

And the following command would copy `GettysburgAddress.txt` from my rlogin account to my current directory on my machine:

```
scp  wmcquain@rlogin.cs.vt.edu:documents/GettysburgAddress.txt  .
```

Identifying a Path Command: which

If you're not sure where a command resides, the **which** command will tell you:


williammcquain@MSI-Ubuntu: ~/Documents
File Edit View Search Terminal Help

```
williammcquain@MSI-Ubuntu:~/Documents$ which ls
/bin/ls
williammcquain@MSI-Ubuntu:~/Documents$ which gcc
/usr/bin/gcc
williammcquain@MSI-Ubuntu:~/Documents$ gcc --version
gcc (Ubuntu/Linaro 4.5.2-8ubuntu4) 4.5.2
Copyright (C) 2010 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

williammcquain@MSI-Ubuntu:~/Documents$
```

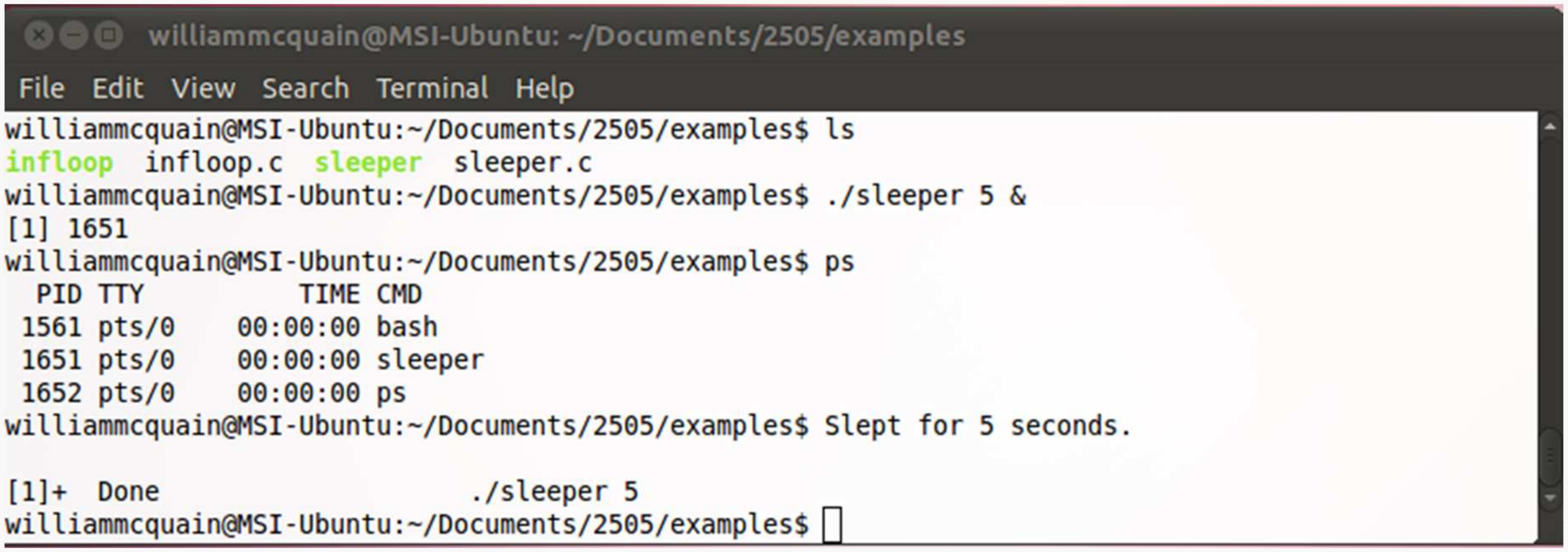
Many Linux applications also support a **--version** switch which can help identify which specific version of an application you're invoking.

Foreground vs Background

By default when you execute a command in a shell, the shell program waits (doesn't provide a prompt and allow entry of another command) until the current command completes (or is otherwise interrupted).

We way the command is running in the *foreground*.

You can modify this behavior and run a command in the *background*:

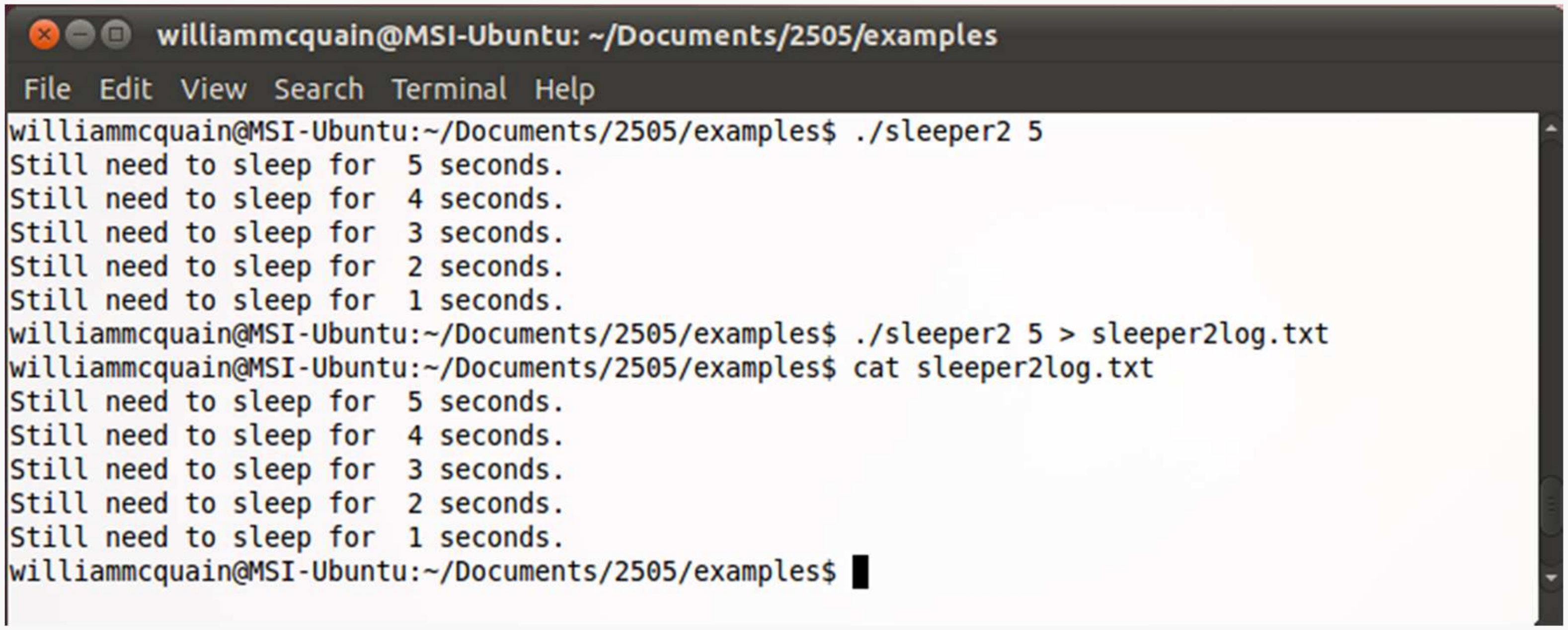

williammcquain@MSI-Ubuntu: ~/Documents/2505/examples
File Edit View Search Terminal Help

```
williammcquain@MSI-Ubuntu:~/Documents/2505/examples$ ls
infloop  infloop.c  sleeper  sleeper.c
williammcquain@MSI-Ubuntu:~/Documents/2505/examples$ ./sleeper 5 &
[1] 1651
williammcquain@MSI-Ubuntu:~/Documents/2505/examples$ ps
  PID TTY          TIME CMD
 1561 pts/0    00:00:00 bash
 1651 pts/0    00:00:00 sleeper
 1652 pts/0    00:00:00 ps
williammcquain@MSI-Ubuntu:~/Documents/2505/examples$ Slept for 5 seconds.

[1]+  Done                    ./sleeper 5
williammcquain@MSI-Ubuntu:~/Documents/2505/examples$ 
```

Redirecting stdout

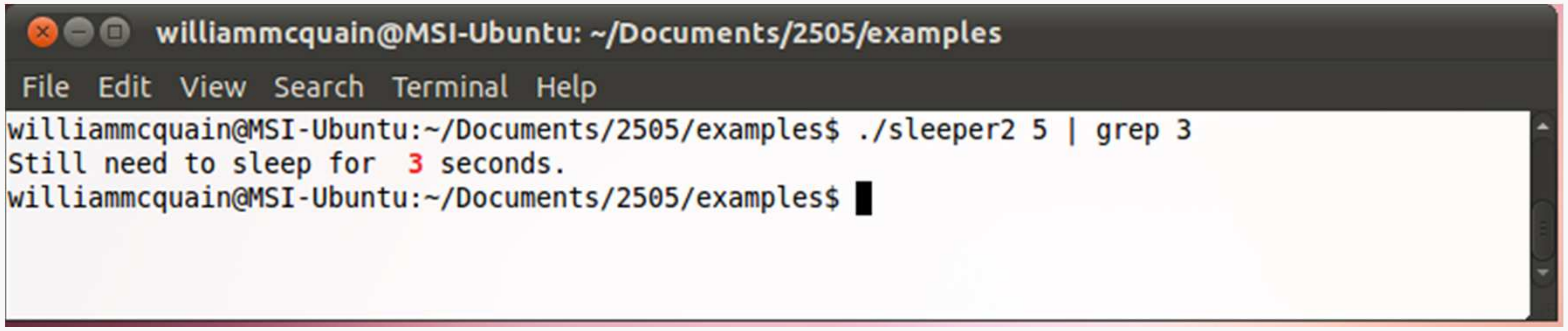
If a process writes output to stdout (the console window), you can *redirect* that into a file:


williammcquain@MSI-Ubuntu: ~/Documents/2505/examples
File Edit View Search Terminal Help

```
williammcquain@MSI-Ubuntu:~/Documents/2505/examples$ ./sleeper2 5
Still need to sleep for  5 seconds.
Still need to sleep for  4 seconds.
Still need to sleep for  3 seconds.
Still need to sleep for  2 seconds.
Still need to sleep for  1 seconds.
williammcquain@MSI-Ubuntu:~/Documents/2505/examples$ ./sleeper2 5 > sleeper2log.txt
williammcquain@MSI-Ubuntu:~/Documents/2505/examples$ cat sleeper2log.txt
Still need to sleep for  5 seconds.
Still need to sleep for  4 seconds.
Still need to sleep for  3 seconds.
Still need to sleep for  2 seconds.
Still need to sleep for  1 seconds.
williammcquain@MSI-Ubuntu:~/Documents/2505/examples$
```

Piping stdout

You can use the *pipe operator* to channel the output from one process as input to another process:


williammcquain@MSI-Ubuntu: ~/Documents/2505/examples
File Edit View Search Terminal Help

```
williammcquain@MSI-Ubuntu:~/Documents/2505/examples$ ./sleeper2 5 | grep 3
Still need to sleep for  3 seconds.
williammcquain@MSI-Ubuntu:~/Documents/2505/examples$ 
```

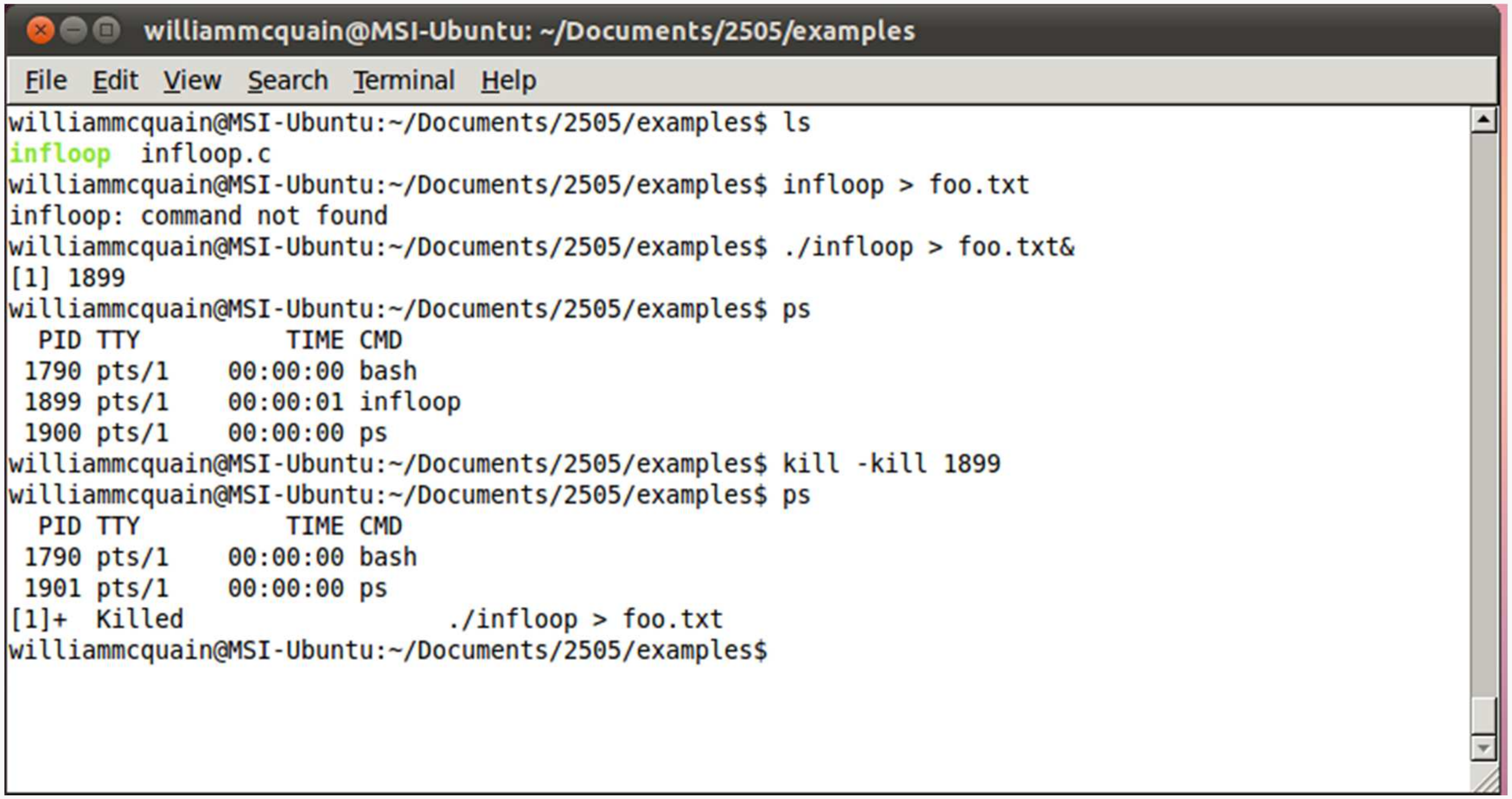
What do you think the following command would do?

./sleeper 5 | grep 3 > filtered.txt

Killing a Process

A (foreground) running process can be killed by using Ctrl-C.

A (background) running process or a suspended process can be killed by using the **kill** command:


williammcquain@MSI-Ubuntu: ~/Documents/2505/examples
File Edit View Search Terminal Help

```
williammcquain@MSI-Ubuntu:~/Documents/2505/examples$ ls
infloop  infloop.c
williammcquain@MSI-Ubuntu:~/Documents/2505/examples$ infloop > foo.txt
infloop: command not found
williammcquain@MSI-Ubuntu:~/Documents/2505/examples$ ./infloop > foo.txt&
[1] 1899
williammcquain@MSI-Ubuntu:~/Documents/2505/examples$ ps
  PID TTY          TIME CMD
 1790 pts/1    00:00:00 bash
 1899 pts/1    00:00:01 infloop
 1900 pts/1    00:00:00 ps
williammcquain@MSI-Ubuntu:~/Documents/2505/examples$ kill -kill 1899
williammcquain@MSI-Ubuntu:~/Documents/2505/examples$ ps
  PID TTY          TIME CMD
 1790 pts/1    00:00:00 bash
 1901 pts/1    00:00:00 ps
[1]+  Killed                  ./infloop > foo.txt
williammcquain@MSI-Ubuntu:~/Documents/2505/examples$
```

Editing Text Files: vi/vim

Editing a text file on your Linux system usually means choosing among:

vi/vim

- the traditional UNIX editor
- complex, somewhat mnemonic “interface”
- a good cheat sheet is essential
- See Chapter 6 in Sobell

gvim

- **vi/vim** with a mouse-aware GUI

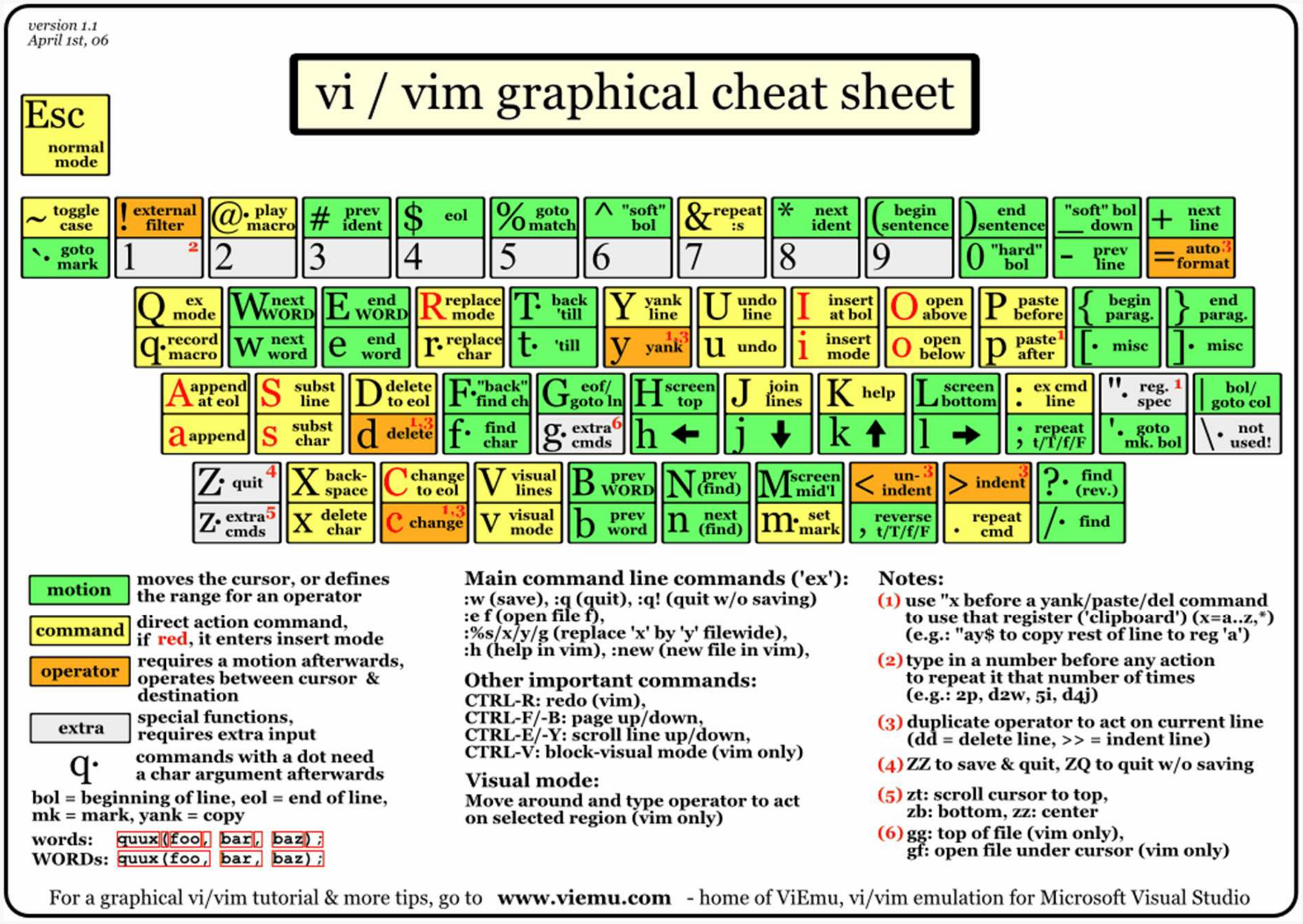
emacs

- a religious experience… sort of like the Aztecs practiced
- See Chapter 7 in Sobell

gedit

- Linux standard text editor
- better than Notepad (well, of course)
- not as full-featured as Notepad++

Editing Text Files: vi/vim



🟩 motion	moves the cursor, or defines the range for an operator
🟨 command	direct action command, if red, it enters insert mode
🟧 operator	requires a motion afterwards, operates between cursor & destination
⬜ extra	special functions, requires extra input
q·	commands with a dot need a char argument afterwards

bol = beginning of line, eol = end of line,
mk = mark, yank = copy

words: quux(foo, bar, baz);
WORDs: quux(foo, bar, baz);

Main command line commands ('ex'):
:w (save), :q (quit), :q! (quit w/o saving)
:e f (open file f),
:%s/x/y/g (replace 'x' by 'y' filewide),
:h (help in vim), :new (new file in vim),

Other important commands:
CTRL-R: redo (vim),
CTRL-F/-B: page up/down,
CTRL-E/-Y: scroll line up/down,
CTRL-V: block-visual mode (vim only)

Visual mode:
Move around and type operator to act on selected region (vim only)

Notes:
(1) use "x before a yank/paste/del command to use that register ('clipboard') (x=a..z,*) (e.g.: "ay$ to copy rest of line to reg 'a')
(2) type in a number before any action to repeat it that number of times (e.g.: 2p, d2w, 5i, d4j)
(3) duplicate operator to act on current line (dd = delete line, >> = indent line)
(4) ZZ to save & quit, ZQ to quit w/o saving
(5) zt: scroll cursor to top, zb: bottom, zz: center
(6) gg: top of file (vim only), gf: open file under cursor (vim only)

For a graphical vi/vim tutorial & more tips, go to **www.viemu.com** - home of ViEmu, vi/vim emulation for Microsoft Visual Studio