GACRC

Introduction to Linux Basics Part I

Georgia Advanced Computing Resource Center

University of Georgia

Suchitra Pakala

pakala@uga.edu

OVERVIEW

- GACRC
- Introduction
- History
- Why Linux?
- How does Linux work?

Georgia Advanced Computing Resource Center

Who Are We:

- **G**eorgia **A**dvanced **C**omputing **R**esource **C**enter (**GACRC**)
- Collaboration between the Office of Vice President for Research (**OVPR**) and the Office of the Vice President for Information Technology (**OVPIT**)
- Guided by a faculty advisory committee (GACRC-AC)

Why Are We Here?

- To provide computing hardware and network infrastructure in support of high-performance computing (**HPC**) at UGA

Where Are We?

- http://gacrc.uga.edu (Web)
- http://wiki.gacrc.uga.edu (Wiki)
- http://gacrc.uga.edu/help/ (Web Help)
- https://wiki.gacrc.uga.edu/wiki/Getting_Help (Wiki Help)

Introduction

- The Linux operating system is an extremely versatile operating system, and has taken a clear lead in the high performance and scientific computing community.
- Nearly 92% of the computers found on the Top500 list run some type of Linux or Unix operating system.
- Multi-user, Multi-tasking operating system
- Open Source
- There are several distributions of Linux.
- Some examples include Ubuntu, CentOS, etc

HISTORY

- The Unix operating system got it's start in 1969 at Bell Laboratories
- The creation of a portable operating system was very significant in the computing industry, but then came the problem of licensing each type of Unix.
- Richard Stallman, an American software freedom activist and programmer recognized a need for open source solutions and launched the GNU project in 1983, later founding the Free Software Foundation.
- In September of 1991 Linus Torvalds released the first version (0.1) of what was to become the Linux kernel. Torvalds greatly enhanced the open source community by releasing his license under the GNU license so that everyone has access to the source code and can freely make modifications to it.

WHY LINUX?

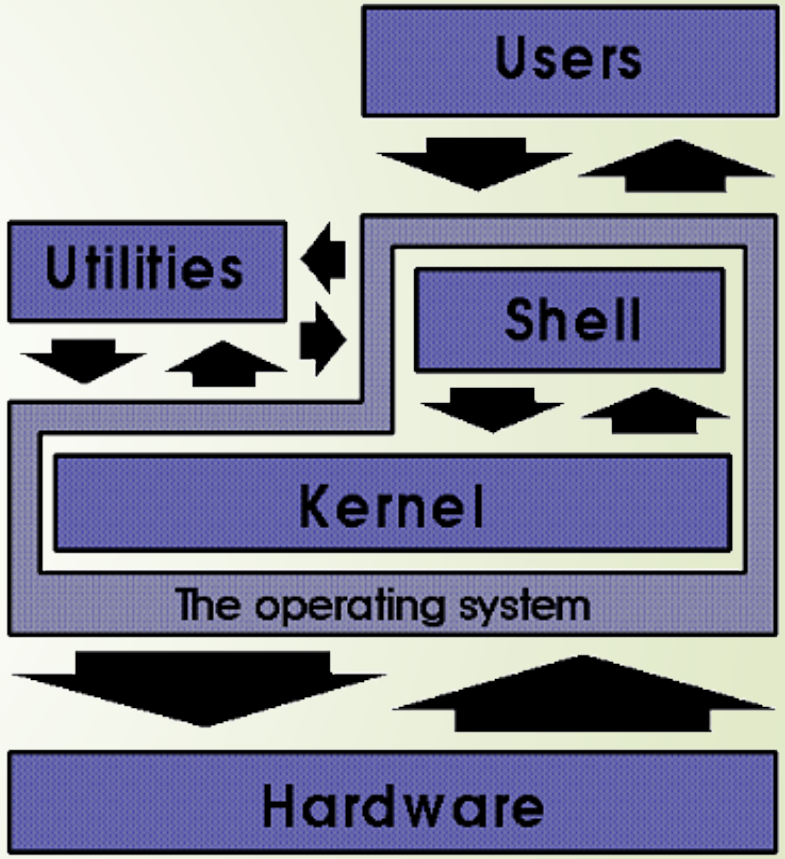
- Linux is free
- Linux systems are extremely stable
- No/very few threats of viruses
- Linux is portable to any hardware platform
- Linux is secure and versatile
- Linux is scalable
- Linux applications have very short debug-times
- Linux comes with most of the required software pre-installed
- Update all your software with minimum fuss

HOW DOES LINUX WORK?

- Linux operating system
- shells
- Files and Processes
- File Permissions
  - Changing File Permissions
- working with shells
  - List Files, Change Directory, Copy, Move, Delete
- Editing Files
- File Conversion

Linux Operating System

- There are two major components of Linux, the kernel and the shell
- The kernel is the core of the Linux operating system which schedules processes and interfaces directly with the hardware
- It manages system and user I/O, processes, devices, files, and memory
- The shell is an interface to the kernel.
- Users input commands through the shell, and the kernel receives the tasks from the shell and performs them
- The shell tends to do four jobs repeatedly: display a prompt, read a command, process the given command, then execute the command
- After which it starts the process all over again



Shells

- "Shell" is a command line interpreter and environment available for user interaction. It executes commands from user via keyboard or a file
- There are several different shells available, each with pros and cons. Different features are supported by different shells. Examples of features: Command-line completion, Command history, mandatory argument prompt, automatic suggestions, auto-correction, etc
- Bash-shell (bash) is the most common one. Other examples: tcsh, ksh
- To determine which shell you are currently using, type the ***echo*** command followed by the system environment variable ***$SHELL***

```
$ echo $SHELL
/bin/bash
```

Files And Processors

- Everything in Linux is considered to be either a ***file*** or a ***process***
- A process is an executing program identified by a unique process identifier, called a **PID**.
- Processes may be short in duration, such as a process that prints a file to the screen, or they may run indefinitely, such as a monitor program
- A file is simply a collection of data, with a location in the file system called a **path**.
- Files can be created by users via **text editors**, or compilers
- The Linux kernel is responsible for organizing processes and interacting with files; it allocates time and memory to each process and handles the file system and communications in response to system calls

Working With Shells

- ***$SHELL*** environment variable, which stores the pathname of the current shell

```
$ echo $SHELL
/bin/bash
```

- ***cat*** is a standard Linux utility that concatenates and prints the content of a file to standard output
- *shells* is the name of the file, and */etc/* is the pathname of the directory where this file is stored

```
$ cat /etc/shells
/bin/sh
/bin/bash
/sbin/nologin
/bin/dash
/bin/tcsh
/bin/csh
/usr/bin/tmux
```

Working With Shells

- To print the current date and time : ***date***

```
$ date
Thu Aug 25 15:05:10 EDT 2016
```

- To list all of your current running processes: ***ps*** command.
- In Linux, each process is associated with a *process identification* (PID)

```
$ ps
  PID TTY          TIME CMD
 9163 pts/6    00:00:00 bash
12194 pts/6    00:00:00 ps
```

Manual Pages

- Linux includes a built in manual for nearly all commands
- The syntax for accessing these manuals is to use the **man** command followed by the program name
- "man" formats and displays the on-line manual pages
- If you specify a section, "man" only displays that section of the manual
- The manual pages follow a common layout. Sections may include the following topics:
  - Name--a one line description of what it does
  - Synopsis--basic syntax for the command line
  - Description--describes the program's functionalities
  - Options--lists command line options that are available for this program
  - Examples--examples of some of the options available

Example: *rm*(Remove).


```
$ man rm
RM(1)                          User Commands                          RM(1)

NAME
    rm - remove files or directories

SYNOPSIS
    rm [OPTION]... FILE...

DESCRIPTION
    This  manual page documents the GNU version of rm.  rm removes each
    specified file. By default, it does not remove directories.

     If the -I or --interactive=once option is given,  and  there  are  more
     than  three  files  or  the  -r,  -R, or --recursive are given, then rm
     prompts the user for whether to proceed with the entire operation.   If
     the response is not affirmative, the entire command is aborted.
```

- Depending on the command, the OPTIONS section can be quite lengthy

```
OPTIONS
    Remove (unlink) the FILE(s).

       -f, --force
              ignore nonexistent files, never prompt

       -i     prompt before every removal

       -r, -R, --recursive
              remove directories and their contents recursively

       -v, --verbose
              explain what is being done
```

Files And File Names

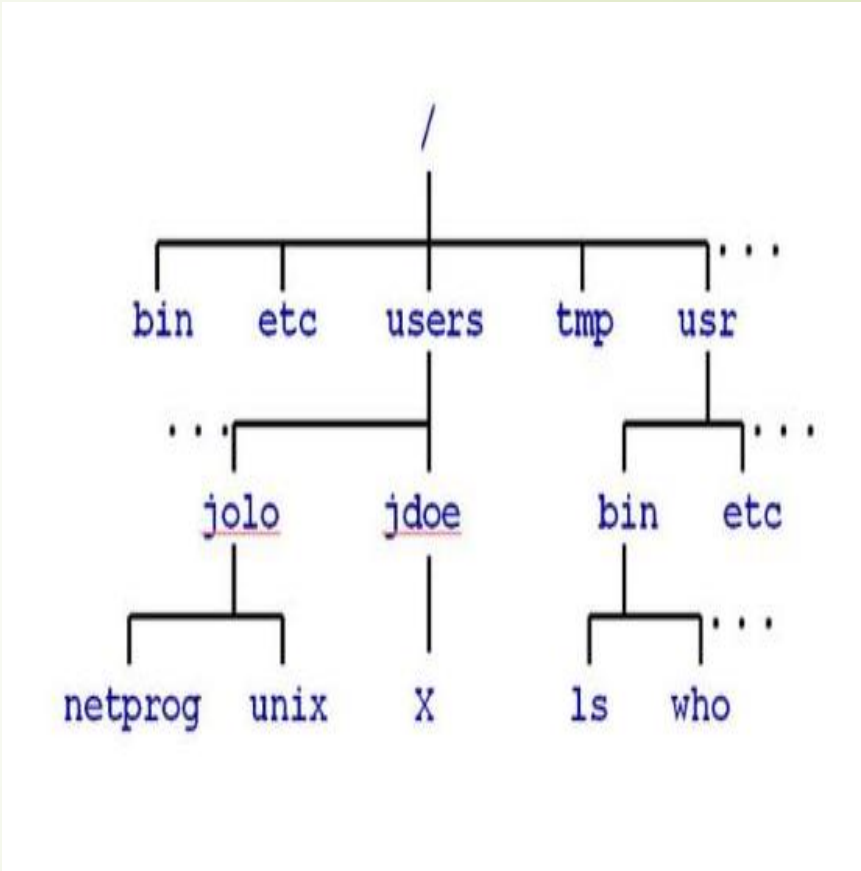
- A file is the basic unit of storage for data
- Every file must have a name as the operating system identifies files by its name
- File names may contain any characters
- You should **avoid spaces, quotes, and parenthesis**
- File names can be long and descriptive, up to 255 characters

- A directory is a special type of file
- Linux uses a directory to hold information about other files
- A directory as a container that holds other files or directories
- The working directory is the directory where you are currently working
- When you first login to a Linux system, your working directory will be your home directory.
- To view which directory you are currently in, use **pwd** command, which displays the **present working directory**

```
$ pwd
/home/gacrc-instruction/pakala
```

File Structure

- In Linux the directory structure is an 'upside down tree'
- The top level directory in any Linux system is called the **root directory** represented by the forward slash /
- All directories are organized inside the root directory
- Users can create directories inside of directories--these are called **sub directories**
- Each file has a name which has to be unique in its containing directory
- Files in different directories can have the same name, but they are distinguished by different directories
  - For example, one could have a file named file1 in the folder /users/jolo/ and another file named file1in the directory /users/jolo/unix/



File Permissions

- Linux is a multi-user environment where many users run programs and share data
- File permissions are used to protect users and system files
- Files and directories have three levels of permissions: **User, Group and World**.
- The types of permissions a file can have are:

Read Permissions	Write Permissions	Execute Permissions
r	w	x

- File permissions are arranged into three groups of three characters each.
- The first set is the User (owner) permissions; the second set is the Group permissions; and finally permissions for Others or everyone else on the system.
- In the following example, the owner can read and write the file, while group and all others have read access only

User (owner)	Group	Others (everyone else)
rw-	r--	r--

File Permissions

- To view a files permissions, "**long list**" option (-l) with ls can be used.

```
$ ls -l
drwxrwxrwx 3 pakala gacrc-instruction 4096 Jul 12 14:40 Blast
drwxrwxr-- 4 pakala gacrc-instruction 4096 Mar  3 12:57 ncbidb
drwxrwxr-- 3 pakala gacrc-instruction 4096 Nov 16  2015 RNA_SEQ
```

- In the above example, user "pakala" owns all files
- Blast directory has read, write and execute access to group and others
- Whereas ncbidb and RNA_SEQ directory only have read, write and execute permissions for group, but others on the system have read access only

Changing File Permissions(chmod)

- **chmod** command to change permissions of a file
  - **Chmod in symbolic mode**

- The syntax of the command in symbolic mode:
  chmod [references][operator][modes]: references can be "u" for user, "g" for group, "o" for others and "a" for all three types
- The operator can be "+" to add and "-" to remove permissions
- In the following example, the owner has been given read, write, and execute permissions, the group and everybody else has no permission

```
$ chmod u+rwx myfile1

$ ls -l myfile1
-rwx------. 1 jdoe community_group 355 2016-02-18 15:50 myfile1
```

Changing File Permissions(chmod)

- **Chmod in numeric mode:**

- The numeric mode is from one to four octal digits (0-7)
- The value for each digit is derived by adding up the bits with **values 4 (read only), 2 (write only), and 1 (execute only)**
- The value **zero removes** all permission for the particular group, whereas the value **7 turns on all permissions** (read, write, and execute) for that group

- Initially **myfile1** is set to read and write for user, and permissions for the group and everybody else is set to read only

```
$ ls -l myfile1
rw-r--r-- 1 jdoe community_group  355 2016-08-25 15:50 myfile1
```

- To change these permissions so that I have read, write, and execute permissions (7), the group has read and execute permisson (5), and everybody else has no permissions (0)

```
$ chmod 750 myfile1
$ ls -l myfile1
-rwxr-x--- 1 jdoe community_group 355 2016-08-25 15:50 myfile1
```

Common Linux Commands

- **cd** : Change your current working directory
- **pwd** : Print absolute path of your current working directory
- **mkdir** : Create a directory
- **rmdir** : Delete an empty directory
- **rm –r** : Delete a nonempty directory and its contents
- **ls** : List the files that exist in the current directory
- **mv** : moves a file to another location.
- **cp** : copies files or directories

Current Directory (cd)

- **cd** will change your current working directory to a new location, given a path.
- For example, to move to the bin subdirectory of the usr directory:

```
$ cd /usr/bin
```

- To move up one directory, to the parent directory of the current working directory

```
$ cd ..
```

- cd command with no arguments, the default action is to return to your home directory

```
$ cd
```

- tilde ~ user name notation or the $HOME environment variable

```
$ cd ~pakala
```

- For example, if my username is pakala, to return to my home directory

```
$ cd $HOME
```

- $HOME is an environment variable which contains the path to your home directory

List Directory(ls)

- **ls**, list the files that exist in the current directory
- ls with **no option** list files and directories in bare format where we won't be able to view details like file types, size, modified date and time, permission and links etc.

```
pakala@zcluster:/escratch4/pakala$ ls
pakala_Feb_02  pakala_Jul_07  pakala_Mar_03
```

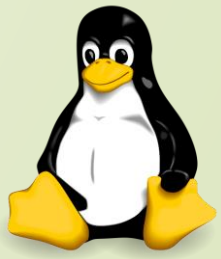
- **ls –l**; shows file or directory, size, modified date and time, file or folder name and owner of file and it's permission

```
pakala@zcluster:/escratch4/pakala$ ls -l
total 9
drwxrwxrwx 3 pakala gacrc-instruction 3 Jul 20 10:24 pakala_Feb_02
drwx------ 3 pakala gacrc-instruction 3 Jul 20 10:29 pakala_Jul_07
drwxrwxrwx 2 pakala gacrc-instruction 2 Mar  3 11:24 pakala_Mar_03
```

List Directory(ls)

- List all files including hidden file starting with '.'

```
 $ ls -a
.bash_history.compute-14-9    .bash_history.copy2      .bashrc  file2.sh    ncbidb
sampledata.sh     .viminfo
.bash_history.compute-18-12   .bash_history.zcluster   Blast    first.sh    RNA_SEQ
samplescript.sh   whileloop.sh
.bash_history.compute-18-16   .bash_history.zhead      car.sh   forloop.sh  R_Program
.ssh
.bash_history.compute-13-21   .bash_history.compute-18-4    .bash_logout     e6
.java         sample      test1.sh
.bash_history.compute-14-7    .bash_history.compute-18-8    .bash_profile    .emacs
.mozilla      sample1     .toprc
```



- With combination of **-lh** option, shows sizes in human readable format

```
$ ls -lh
drwxrwxrwx 3 pakala gacrc-instruction 4.0K Jul 12 14:40 Blast
-rwxr-xr-x 1 pakala gacrc-instruction  497 Aug 26 11:22 car.sh
lrwxrwxrwx 1 pakala gacrc-instruction   31 Jul 12 09:19 e6 ->
/escratch4/pakala/pakala_Jul_07
-rwxr-xr-x 1 pakala gacrc-instruction  322 Aug 26 03:47 whileloop.sh
```

- **lS** displays file size in order, will display big in size first

```
pakala@zcluster:~$ ls -lS
total 960
drwxrwxrwx 3 pakala gacrc-instruction 4096 Jul 12 14:40 Blast
drwxr-xr-x 4 pakala gacrc-instruction 4096 Mar  3  2016 ncbidb
-rwxr-xr-x 1 pakala gacrc-instruction  497 Aug 26 11:22 car.sh
-rwxr-xr-x 1 pakala gacrc-instruction  322 Aug 26 03:47 whileloop.sh
lrwxrwxrwx 1 pakala gacrc-instruction   31 Jul 12 09:19 e6 ->
/escratch4/pakala/pakala_Jul_07
```

Move Files(mv)

- **mv** moves a file to another location.

- For example, to move a file from /users/jolo/netprog to /users/jolo/unix:

```
$ mv /users/jolo/netprog/myFile /users/jolo/unix
```

- This can also be used to rename a file in the same directory.
- For example, to rename *myFile* to *myFile.old*:

```
$ mv myFile myFile.old
```

- **Other options:**

```
option     description
mv -f      force move by overwriting destination file without prompt
mv -i      interactive prompt before overwrite
mv -u      update - move when source is newer than destination
mv -v      verbose - print source and destination files
man mv     help manual
```

Copy Files(cp)

- **cp** copies files or directories.
- To copy a file from /users/jolo/unix to /users/jolo/netprog:

```
$ cp /users/jolo/unix /users/jolo/netprog/myOtherFile
```

- **Other Options:**

```
option     description
cp -a      archive files
cp -f      force copy by removing the destination file if needed
cp -i      interactive - ask before overwrite
cp -l      link files instead of copy
cp -L      follow symbolic links
cp -n      no file overwrite
cp -R      recursive copy (including hidden files)
cp -u      update - copy when source is newer than dest
cp -v      verbose - print informative messages
```

Remove Files(rm)

- **rm** removes files

```
$ rm /users/jolo/netprog/myOtherFile
```

- **Other options:**

```
option      description
Remove (unlink) the FILE(s)
rm -f         ignore nonexistent files, never prompt
rm -i         prompt before every removal
rm -r, -R     remove directories and their contents recursively
rm -v         explain what is being done
```

- With the **-r** or **-R** option, it will also remove/delete entire directories recursively and permanently.
- **rm -r *** will remove all of the files and subdirectories within your current directory.
- To remove an empty directory, use **rmdir**

Relative Path vs Absolute Path

- The **absolute or full path** is the entire directory structure pointing to a file
- A **relative path** is the path from where you are now (your present working directory) to the file in question
- An easy way to know if a path is absolute is to check if it contains the "/" character at the beginning of the path
- For example, there is a directory in my home directory called Blast
- Since my home directory is **/home/gacrc-instruction/pakala**
- I could list the file using "**ls**" with the **absolute path**:

```
$ ls /home/gacrc-instruction/pakala/Blast/
AF293     Escherichia_Coli_LF82_Chromosome_Sequence.fasta
blast.sh  GCF_000002655.1_ASM265v1_genomic.fna
```

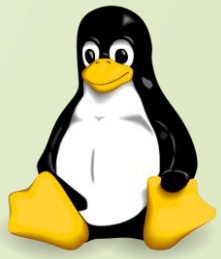
Relative Path vs Absolute Path

- If I was already in my home directory, I could use a **relative path**, which starts from my current working directory:

```
$ pwd
/home/gacrc-instruction/pakala

$ ls Blast
AF293      Escherichia_Coli_LF82_Chromosome_Sequence.fasta
blast.sh   GCF_000002655.1_ASM265v1_genomic.fna
```

- The construct "**./**" explicitly specifies the base path the current working directory
- **pwd** writes the full path of the current working directory



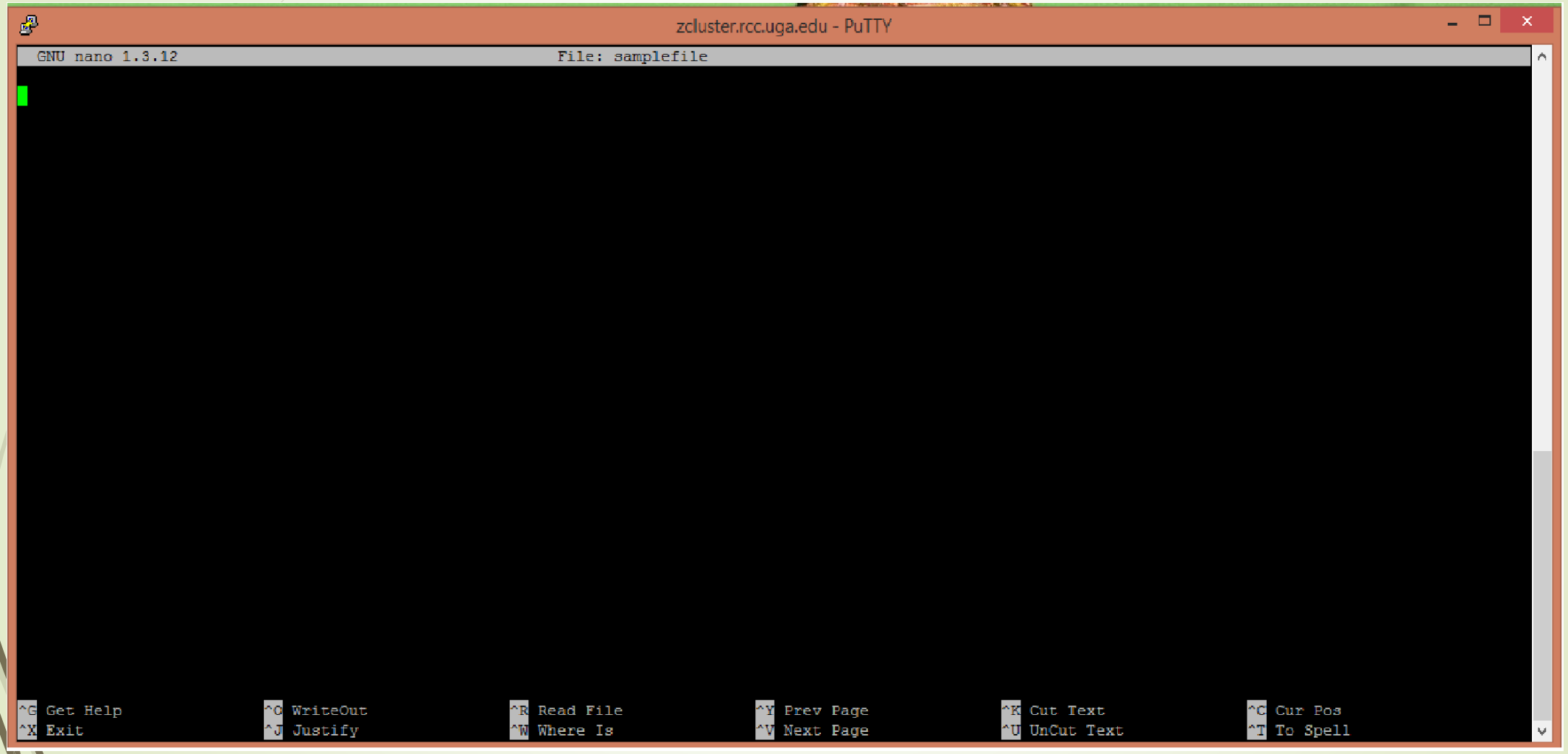
Editing Files

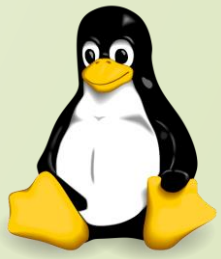
- A text editor is a simple tool to assist the user with creating and editing files.
- The most widely used editors available on zcluster or sapelo are emacs, vi, nano etc
  - **Creating a File:**
- Type nano followed by the file name you want to create and edit

```
$ nano text.tmp
```

- At the top, you'll see the name of the program and version number, the name of the file you're editing, and whether the file has been modified since it was last saved.
- If you have a new file that isn't saved yet, you'll see "New Buffer."
- Next, you'll see the contents of your document, a body of text.
- The third-line from the bottom is a "system message" line that displays information relevant to the program executing a function.
- Lastly, the final two rows at the bottom are what make this program very user-friendly: the shortcut lines.

Nano Text Editor





Word and Line Count

- The wc command reads either STDIN or a list of files and generates
  - numbers of lines
  - numbers of words
  - numbers of bytes.
- Line count: -l
- Word count: -w
- Byte count: -c

```
$ cat temp.txt
cherry
apple
x-ray
clock
orange
Bananna

$ wc temp.txt
 6  6 40 temp.txt
```

```
$ wc -l temp.txt
6 temp.txt


$ wc -w temp.txt
6 temp.txt


$ wc -c temp.txt
40 temp.txt
```

- There are 6 lines, 6 words, and 40 bytes (or characters) in the file temp.txt

File Conversion

- **dos2unix** : Convert DOS/Windows file to Linux format

- Example: dos2unix file1
- Removes DOS/Windows line endings in file1

```
$ dos2unix file1
```

- **mac2unix** : Convert Mac file to Linux format

- Example: mac2unix file1
- Removes Mac line endings in file1

```
$ mac2unix file1
```

THANK YOU ☺