ITWS -1

Monsoon 2017

Kirti Garg and Lalit Mohan S

kirti@iiit.ac.in, lalit.mohan@iiit.ac.in

Unix Command Line

Why Command Line?

- Why use command line when we have GUI?
  - Achieve complex tasks
  - Get things done quickly and efficiently
  - Perform tasks originally not thought of by authors
  - Automate tasks easily

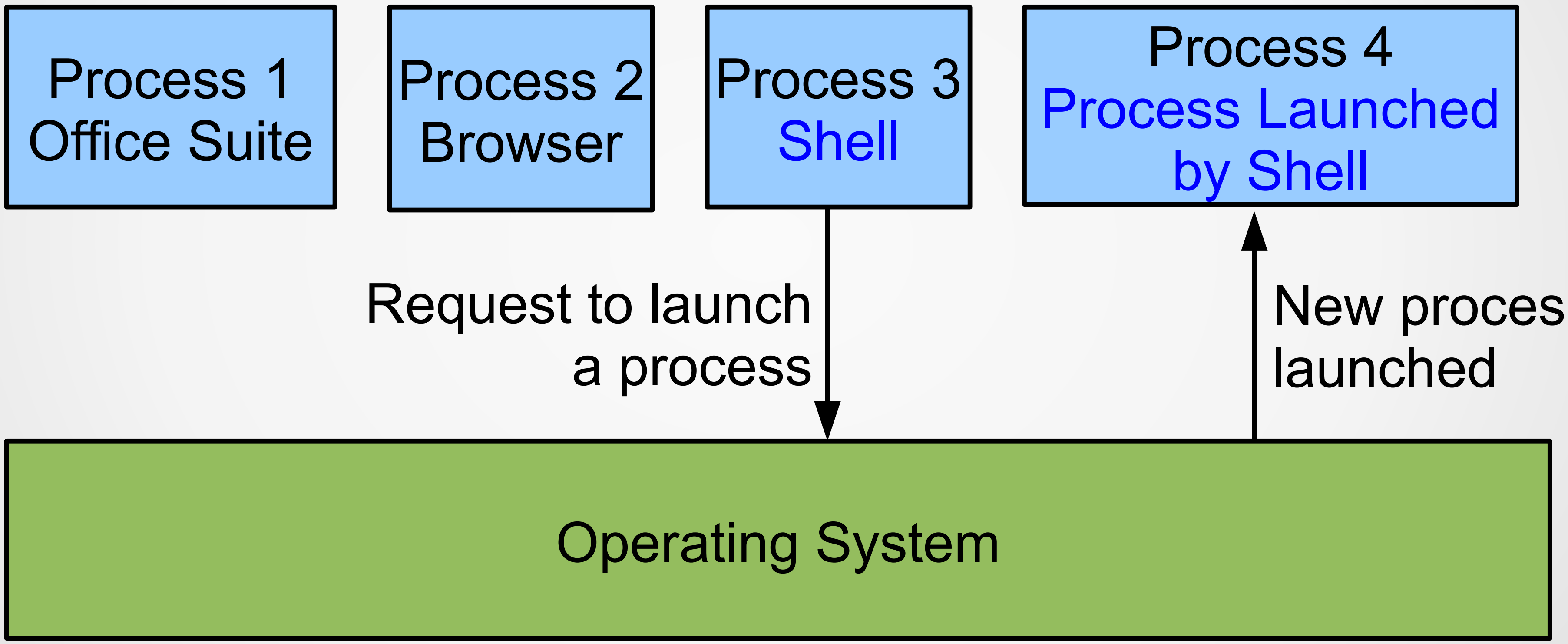
Example Use Cases

- Mr. A wishes to retrieve all files modified last week and replace the phrase “this week” with “next week” in those files.
- Everyday, Ms. B wishes to automatically retrieve all files modified on that day and back them up to different location.
- Ms. C likes to rename files so that their extensions are removed
- Mr. D likes to combine to merge fives sets of user lists into a single one

Unix Philosophy

- Do one thing and do it well
- Programs work together
- Simplicity
- Communicate using text streams

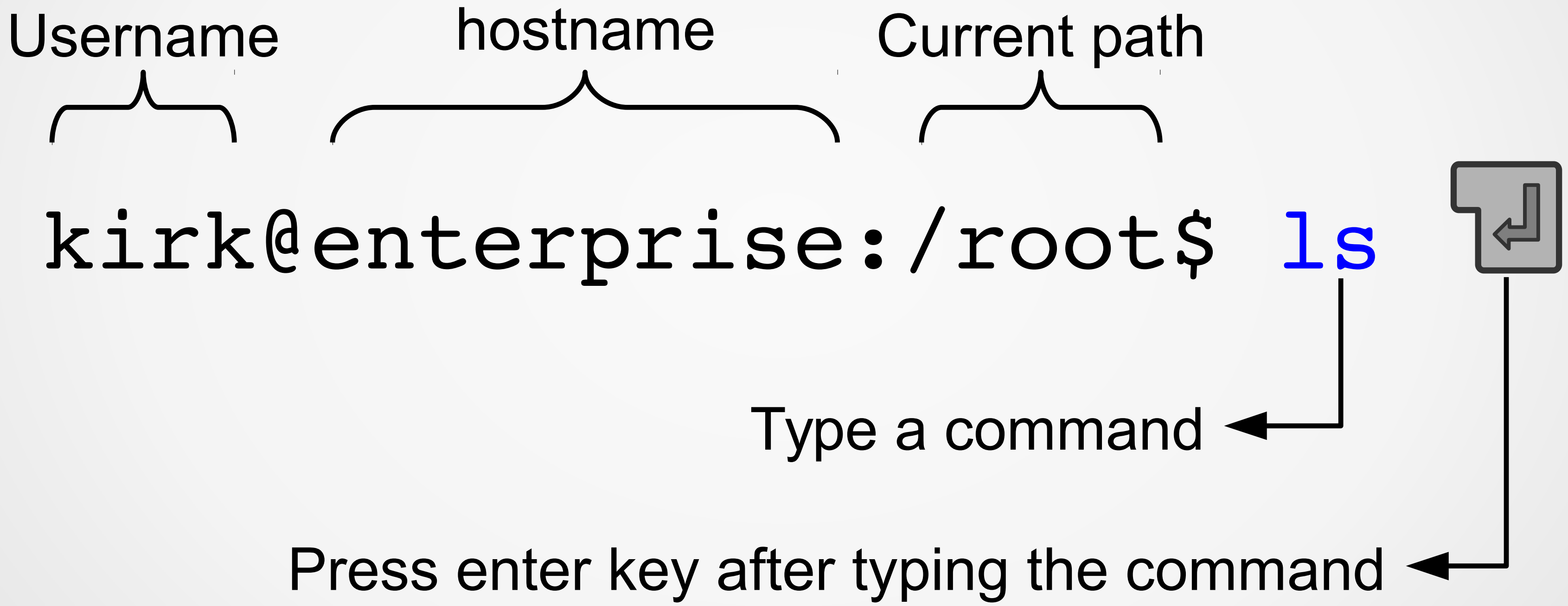
Shell


Process 1
Office Suite
Process 2
Browser
Process 3
Shell
Process 4
Process Launched by Shell
Request to launch a process
New process launched
Operating System

Launching a Shell

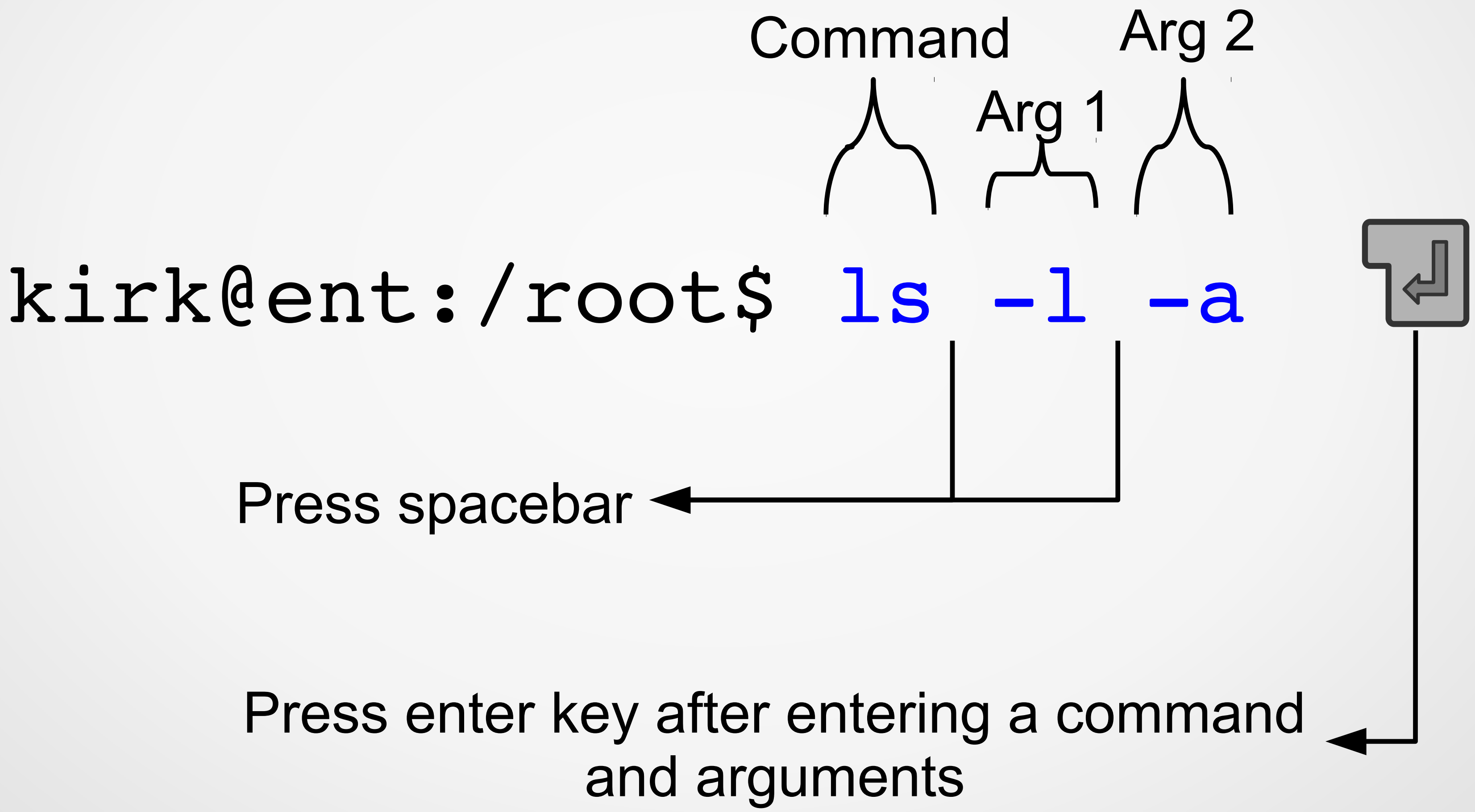
- Press Control-Alt-F1 to go to text console and login
  - Press Control-Alt-F7 to back to graphical mode
- Search for and open an application named “Terminal”
- Login at console as another user
- Remote login into another machine

Issuing a Command at the Prompt



```
kirk@enterprise:/root$ ls
```

Command with Arguments



```
kirk@ent:/root$ ls -l -a
```

ls – List Files & Directories

```
kirk@enterprise:/$ ls
bin    initrd.img  mnt   run   usr
boot   lib         null  sbin  var
dev    lib64       opt   srv   vmlinuz
etc    lost+found  proc  sys
home   media       root  tmp
kirk@enterprise:/$
```

Output of command

Command prompt after completion of 'ls' command

Getting Help on Commands

- man ls to see the 'manual page' of command ls
- info ls to get full documentation of ls
- man man to get help on how to use man command
- info info to get information on how to use the info browser
- Discover new commands by pressing <TAB> <TAB> and by reading their manual pages

Exploring the File System

Paths

$$/\overbrace{\texttt{usr}}^{\text{Directory}}/\overbrace{\texttt{local}}^{\text{Sub-directory}}\overset{\text{Path Separator}}{/}\texttt{bin}$$

- / is the top most directory. It is also the path separator
- . is the current directory
- .. is the parent directory. /home/user/work/.. is same as /home/user
- ~ is the home directory of the current user

Prompt Working Directory

- pwd to show the current directory

Working directory is also shown in the prompt

```
kirk@enterprise:/usr/local$ pwd
/usr/local
```

Change Working Directory

- cd *mypath* to switch to a directory
- cd to switch to home directory

Notice the change in path

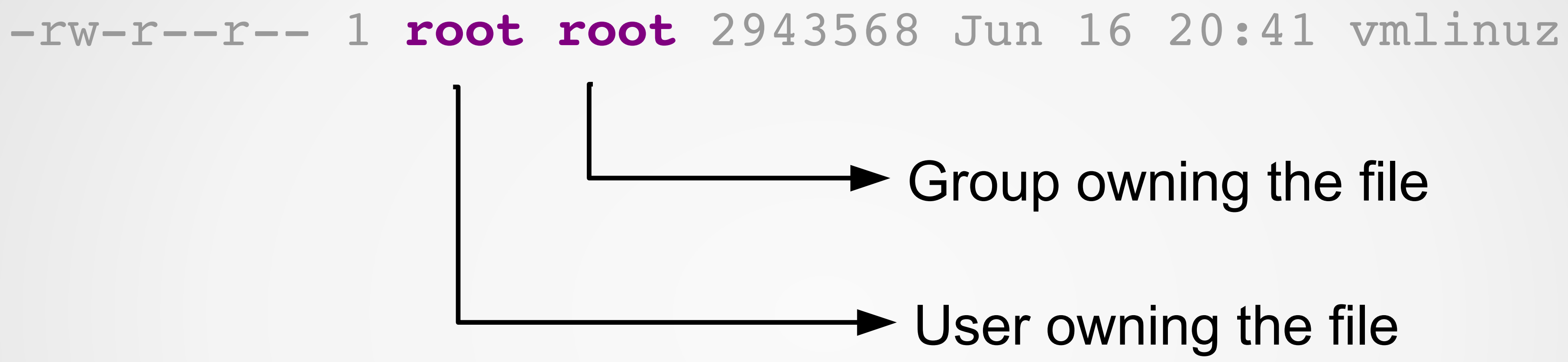
```
kirk@enterprise:/usr$ cd local
kirk@enterprise:/usr/local$ cd /usr/bin
kirk@enterprise:/usr/bin$ cd /
kirk@enterprise:/$ cd ~
kirk@enterprise:/home/kirk$ cd ..
kirk@enterprise:/home$ cd .
kirk@enterprise:/home$
```

Output of “ls -l”

```
kirk@enterprise:/boot$ ls -l
total 14884
-rw-r--r-- 1 root root  153275 Jun 16 20:46 config
drwxr-xr-x 3 root root    4096 Mar 26 10:07 extlinux
drwxr-xr-x 5 root root   12288 Jun 19 01:24 grub
-rw-r--r-- 1 root root 9702279 Jun 18 01:13 initrd.img
-rw-r--r-- 1 root root 2417043 Jun 16 20:46 System.map
-rw-r--r-- 1 root root 2943568 Jun 16 20:41 vmlinuz
```

Permissions   Ownership   Size   Modified time   Name

Ownership

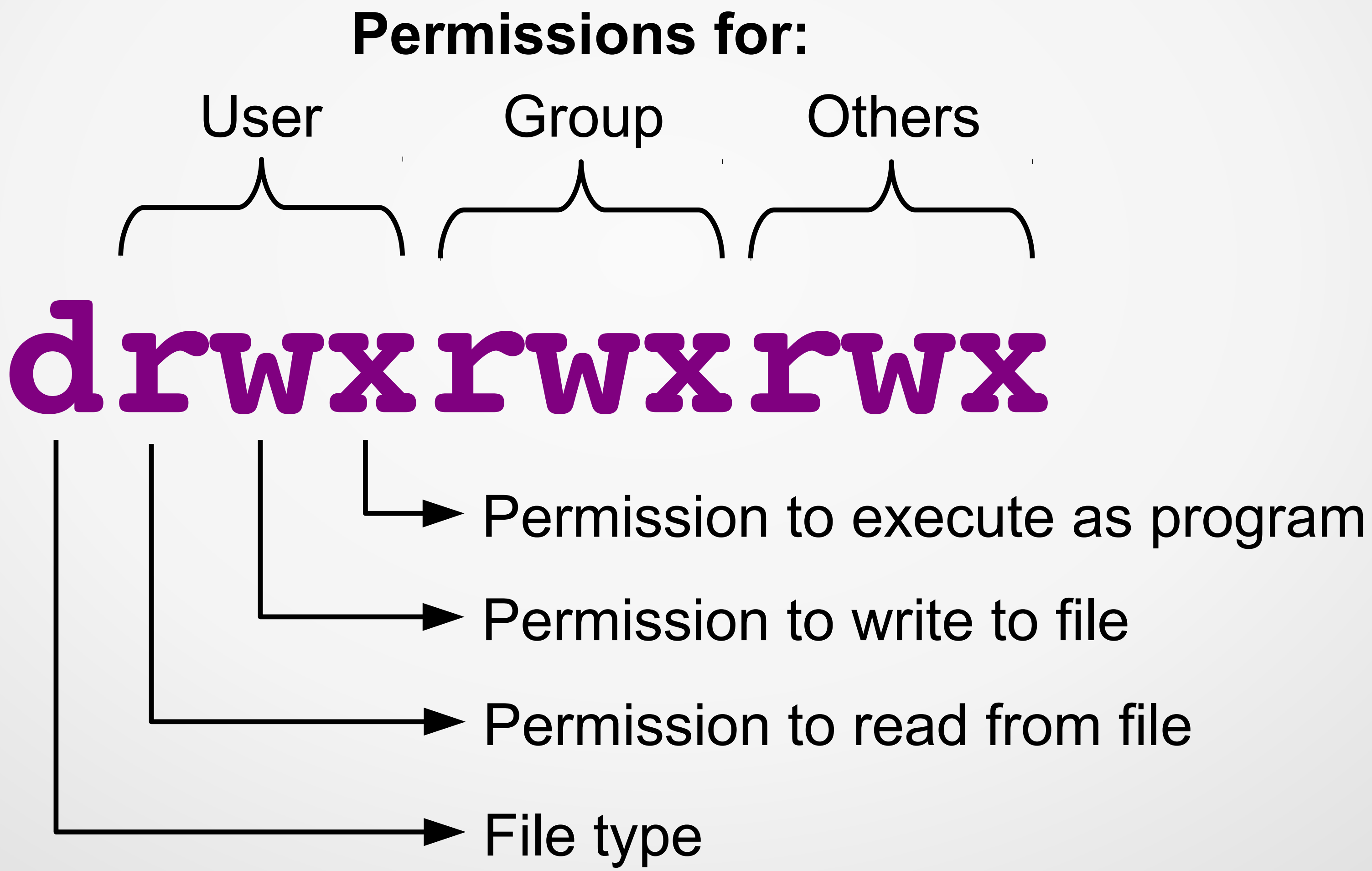


```
-rw-r--r-- 1 root root 2943568 Jun 16 20:41 vmlinuz
```

- A “group” is a set of users
- Each file or directory is owned by one user and one group
- Users are listed in the file /etc/passwd
- Groups are listed in the file /etc/group

Permissions

```
-rw-r--r-- 1 root root 2943568 Jun 16 20:41 vmlinuz
```



Permissions for:
User
Group
Others
drwxrwxrwx
Permission to execute as program
Permission to write to file
Permission to read from file
File type

Example: File Permissions

```
-rw-r----- 1 kirk engineers 2568 Jun 16 20:41 vmlinuz
```

- *Others* can't *execute* the file as a program
- *Others* can't *write* to the file
- *Others* can't *read* the file

- Users in *engineers* group can't *execute* the file
- Users in *engineers* group can't *write* to the file
- Users in *engineers* group can *read* the file

- User *Kirk* can't *execute* the file as a program
- User *Kirk* can *write* to the file
- User *Kirk* can *read* the file

Example: Directory Permissions

```
drwxr-x--- 1 kirk engineers 2568 Jun 16 20:41 vmlinuz
```

Others can't enter and access files within

Others can't create/rename/delete files

Others can't list files in the directory

Users in *engineers* group can enter and access files within

Users in *engineers* group can't create/rename/delete files

Users in *engineers* group can list files in the directory

User *Kirk* can enter and access files within

User *Kirk* can create/rename/delete files

User *Kirk* can list files in the directory

File Types

- - is a regular file
- d is a directory
- l is a symbolic link
- s is a UNIX socket
- b is a block device
- c is a character device
- etc.

Tip: Everything is a file!

- Directories, links, hardware devices, and other communication mechanisms are exposed as files
- Want to read the CDROM contents? Simply read the file representing the CDROM device.
- Want to output to a sound card? Write to the device representing the sound card.

File System Organisation

- /root and /home store user data
- /bin, /usr/bin, /sbin and /usr/sbin store executable commands
- /usr stores files related to user applications
- /usr/local contains applications compiled by the user
- /var contains (variable) files that usually grow over time
- /lib, /usr/lib contains libraries
- /tmp contains temporary files
- /proc is a virtual file system containing kernel information
- /mnt and /mount contain file system mounts

Manipulating Files

Creating a File

- touch *filename* creates an empty file

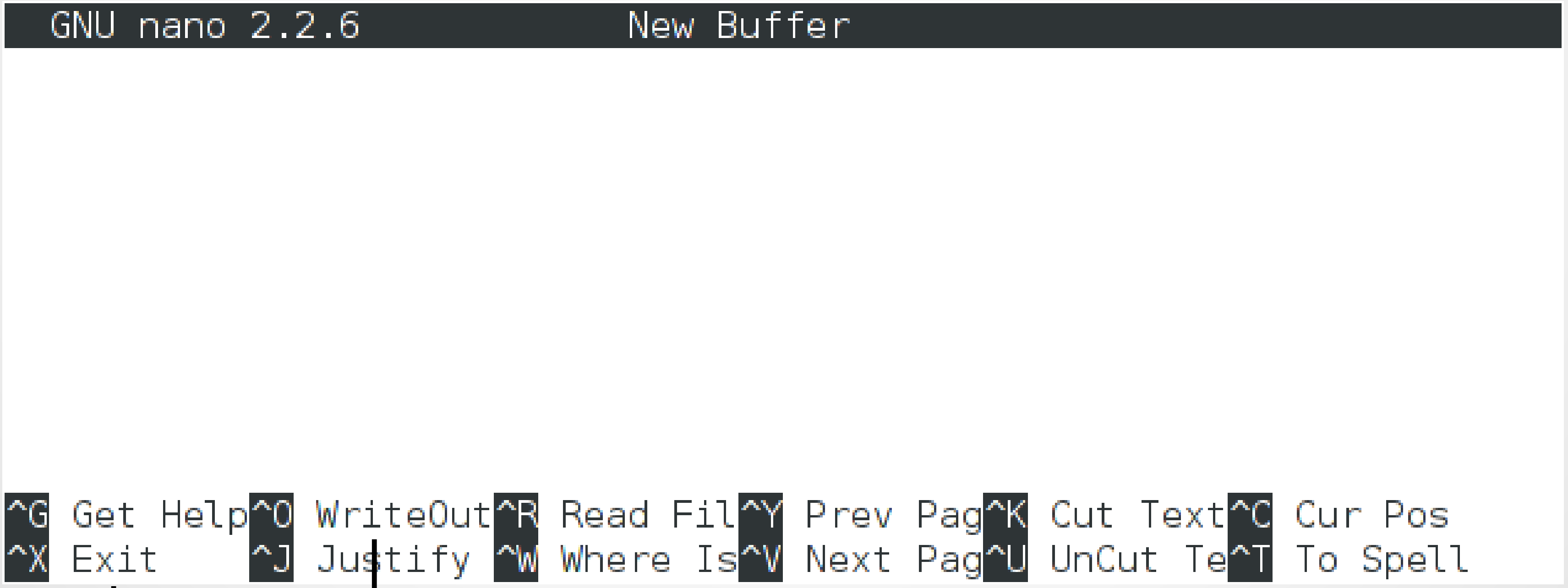
```
kirk@enterprise:/work$ touch new_plan
kirk@enterprise:/work$ ls -l new_plan
-rw-r--r-- 1 kirk kirk 0 Aug  7 08:09 new_plan
```

$$\downarrow$$

File is empty

Simple Text Editor

- nano is a simple text editor
- nano *filename* to edit an existing file



```
  GNU nano 2.2.6                New Buffer









^G Get Help^O WriteOut^R Read Fil^Y Prev Pag^K Cut Text^C Cur Pos
^X Exit    ^J Justify ^W Where Is^V Next Pag^U UnCut Te^T To Spell
```

Press Control-O to save

Press Control-X to exit

Displaying the Contents of a File

- cat *filename* shows the contents of a file.

```
kirk@enterprise:~/work$ cat hello.txt
Hello, World!
kirk@enterprise:~/work$
```

File contents

Creating a File Using Redirection

- cat > *filename* creates a file with typed in content. Press Control-D after you are done entering text.

```
kirk@enterprise:~/work$ cat > flight_plan
To Vulcan
Then to Kronos
{Control-D}
kirk@enterprise:~/work$ cat flight_plan
To Vulcan
Then to Kronos
kirk@enterprise:~/work$
```

Copying Files

- cp *origfile destfile* copies one file to another file

```
kirk@ent:~/work$ cp flight_plan flight_plan_dup
kirk@ent:~/work$ cat flight_plan_dup
To Vulcan
Then to Kronos
kirk@ent:~/work$ ls
flight_plan  flight_plan_dup
```

Creating a Directory

- mkdir *dirname* creates a directory

```
kirk@ent:~/work$ mkdir mydir
kirk@ent:~/work$ ls -l
...
drwxr-xr-x 2 kirk kirk 4096 Aug  7 08:26 mydir
```

Directory created

Copying Multiple Files

- *cp file1 file2 ... dirname* can also copy multiple files to a directory

```
kirk@ent:~/work$ cp flight_plan flight_plan_dup mydir

kirk@ent:~/work$ ls
flight_plan  flight_plan_dup mydir

kirk@ent:~/work$ cd mydir

kirk@ent:~/work/mydir$ ls
flight_plan  flight_plan_dup
```

Rename a File

- mv *origname newname* renames a file

```
kirk@ent:~/work/mydir$ ls
flight_plan  flight_plan_dup

kirk@ent:~/work/mydir$ mv flight_plan fp

kirk@ent:~/work/mydir$ ls
flight_plan_dup  fp
```

Move a File

- *mv filename dirname* also moves a file from one directory to another

```
kirk@ent:~/work/mydir$ ls
fp  flight_plan_dup

kirk@ent:~/work/mydir$ mv fp ..

kirk@ent:~/work/mydir$ ls
flight_plan_dup

kirk@ent:~/work/mydir$ cd ..

kirk@ent:~/work$ ls
flight_plan  flight_plan_dup  fp  mydir
```

Deleting Files

- rm *filename* deletes a file

```
kirk@ent:~/work$ ls
flight_plan  flight_plan_dup  fp  mydir

kirk@ent:~/work$ rm flight_plan_dup

kirk@ent:~/work$ ls
flight_plan  fp  mydir
```

Deleting Directories

- rmdir *dirname* removes an empty directory

```
kirk@ent:~/work$ ls
flight_plan  fp  mydir

kirk@ent:~/work$ rmdir mydir
rmdir: failed to remove ‘mydir/’: Directory not empty

kirk@ent:~/work$ rm mydir/flight_plan_dup

kirk@ent:~/work$ ls mydir

kirk@ent:~/work$ rmdir mydir

kirk@ent:~/work$ ls
flight_plan  fp
```

Deleting Files Recursively

- rm -rf *dirname* removes a directory and its contents recursively

```
kirk@ent:~/work$ mkdir mydir

kirk@ent:~/work$ touch mydir/file1

kirk@ent:~/work$ rmdir mydir
rmdir: failed to remove ‘mydir/’: Directory not empty

kirk@ent:~/work$ rm -rf mydir

kirk@ent:~/work$ ls
flight_plan  fp
```

Tip: You Will Delete Your Data!

- Be careful with the rm and especially rm -rf
- Chances are you will delete important data at least once
- Use rm -i for interactive deleting

Tip: Auto Complete in Shell (Bash)

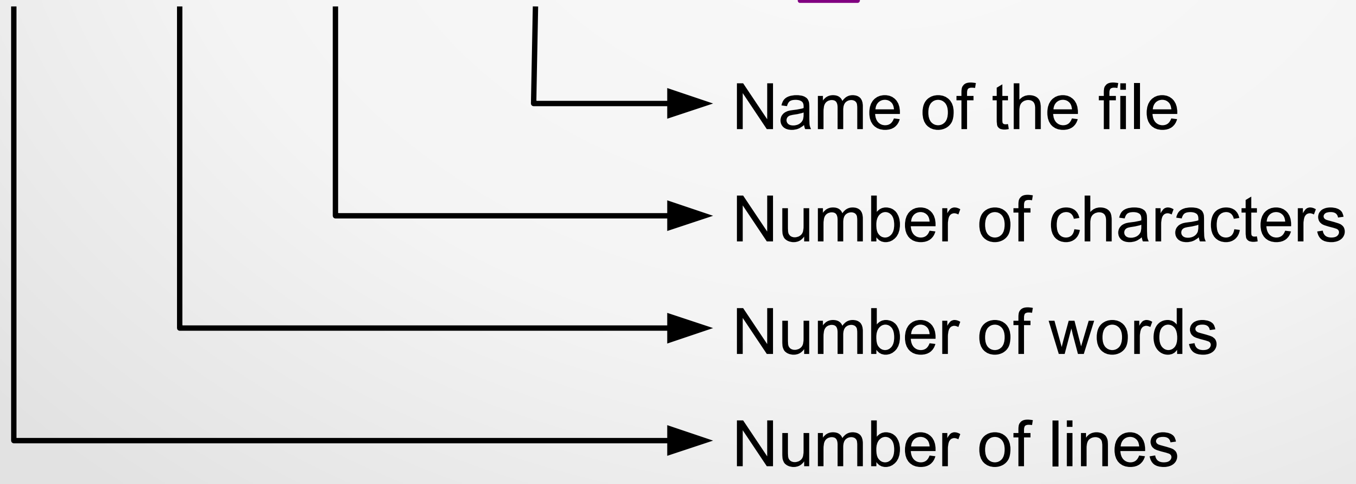
- TAB key to finish a half-entered command
- Up and down arrows to browse command history
- ! to run previous command
  - !*command* will repeat previous invocation of *command*
- history to see the earlier typed commands
- Control-R to search a previously issued command

Text Processing

Word Count

- wc *filename* counts number of characters, words and lines in a file

```
kirk@enterprise:~$ wc flight_plan
 2  5 25 flight_plan
```



Getting the Top Part

- head *filename* prints the top 10 lines of a file
- head -n *x filename* prints the top x lines of a file

```
kirk@ent:~$ cat flight_plan
To Vulcan
Then to Kronos

kirk@ent:~$ head -n 1 flight_plan
To Vulcan
```

Getting the Bottom Part

- tail *filename* prints the bottom 10 lines of a file
- tail -n *x filename* prints the bottom x lines of a file
- tail -f *filename* watches the file continuously and prints it

```
kirk@ent:~$ cat flight_plan
To Vulcan
Then to Kronos

kirk@ent:~$ tail -n 1 flight_plan
Then to Kronos
```

Sort the Contents

- sort *filename* to sort data in a file

```
kirk@ent:~$ cat inventory.txt
Phasers - 10
Replicators – 2
Communicators – 100

kirk@ent:~$ sort inventory.txt
Communicators - 100
Phasers - 10
Replicators – 2
```

Output is sorted alphabetically

Omit Repeated Lines

- uniq *filename* to omit repeated lines
- Typically used after sorting

```
kirk@ent:~$ cat inventory.txt
Communicator
Communicator
Communicator
Phaser

kirk@ent:~$ uniq inventory.txt
Communicator   } Repeating lines omitted
Phaser
```

Cut Sections from Each Line

- cut -d *delim* -f *fields file* cuts given *fields* from a file with fields delimited by *delim*

```
kirk@ent:~$ cat /etc/passwd
...
kirk:x:1000:1000::/home/kirk:/bin/bash
spock:x:1001:1001::/home/spock/:/bin/bash
```

Delimiter is a colon ":"

Requesting field 6

```
kirk@ent:~$ cut -d : -f 6 /etc/passwd
/home/kirk
/home/spock
```

Data in field 6,
Home directories of all the users

Merge Lines of Files

- paste *file1 file2* merges lines in *file1* with *file2*

```
kirk@ent:~$ cat names
kirk
spock
scott
kirk@ent:~$ cat roles
bridge
science
kirk@ent:~$ paste -d : names roles
kirk:bridge
spock:science
scott:
```

Delimiter should be a colon ":"

Individual lines merged

Selecting Matching Lines

- grep *pattern file* prints lines from *file* matching a *pattern*

```
kirk@ent:~$ cat names
kirk
spock
scott
mccoy

kirk@ent:~$ grep "co" names
scott
mccoy
```

} Lines containg text “co” in them

Redirection

Redirect Output to a File

- *command* > *outfile* stores the output of *command* into *outfile*
- *outfile* is completely overwritten

```
kirk@ent:~$ ls
flight_plan inventory
```

Normal "ls" output

Redirection of "ls" output to a file

```
kirk@ent:~$ ls > captured_ouput.txt
```

No output to terminal this time

```
kirk@ent:~$ ls
flight_plan inventory captured_output.txt
```

New file created

```
kirk@ent:~$ cat captured_output.txt
flight_plan inventory captured_output.txt
```

Captured output

Redirect Output to a File (Append)

- *command >> outfile* stores the output of *command* into *outfile*
- *outfile* is appened to instead of overwritten

```
kirk@ent:~$ ls
flight_plan inventory
```

Redirection of “ls” output to a file

```
kirk@ent:~$ ls > captured_ouput.txt
kirk@ent:~$ ls >> captured_ouput.txt

kirk@ent:~$ cat captured_output.txt
flight_plan inventory captured_output.txt
flight_plan inventory captured_output.txt
```

Appended output

Redirect Input from a File

- *command < filename* redirects the contents of *filename* as input to *command*

```
kirk@ent:~$ cat inventory
Communicator
Phaser
Communicator
```

Original file contents

```
kirk@ent:~$ wc -l < inventory
3
```

Inventory file processed by “wc -l”:
Number of lines in the file

Piping

- *command1 | command2* redirects the output of *command1* as input to *command2*

```
kirk@ent:~$ cat inventory
Communicator
Phaser
Communicator
```

Original file contents

```
kirk@ent:~$ sort inventory
Communicator
Communicator
Phaser
```

Sorted file contents

Piping (contd.)

Ouput of “sort inventory” is sent as input to “uniq” because of pipe operator

```
kirk@ent:~$ sort inventory | uniq
Communicator
Phaser
```

Sorted & unique file contents

```
kirk@ent:~$ cat inventory | sort | uniq
Communicator
Phaser
```

Piping among three commands

Paginated Display

- less and more for paginated display

```
kirk@ent:~$ ls /usr | more
bin
games
include
lib
libexec
local
--More--
lib
local
sbin
src
kirk@ent:~$
```

One page of output

“more” waits here until <space> is pressed

Final page of output

“ls” and “more” are completed

Wildcards

* Wildcard

- Shell expands * to all filenames

```
kirk@ent:~$ ls
flight_plan inventory

kirk@ent:~$ cat *
To Vulcan
Then to Khronos
Communicator
Phaser
```

Shell expands this to "cat flight_plan inventory"

Contents of "flight_plan"

Contents of "inventory"

* Wildcard (contd.)

- Shell expands _part*_ to all filenames starting with _part_
- Shell expands _*_ to all filenames ending with _part_

Shell expands this to "cat flight_plan"

```
kirk@ent:~$ cat flight*
To Vulcan
Then to Khronos
```

Contents of "flight_plan"

* Wildcard (contd.)

- Shell expands */path/** to all filenames in */path/*

Shell expands this to all files in path "/usr/local" that end with "bin"

```
kirk@ent:~$ ls -d /usr/local/*bin
/usr/local/bin /usr/local/sbin
```

Other Wildcards

- *?* expands to match a single character in the filename
- *[a-z]* expands to match a single character from *a* to *z*

User Related Commands

Who Am I?

- whoami shows the current user

```
kirk@enterprise:~$ whoami
kirk
```

User ID

- id shows the current user and group information

```
kirk@enterprise:~$ id
uid=1000(bunny) gid=1000(bunny) groups=1000(bunny),
6(disk), 7(lp), 24(cdrom), 25(floppy), 27(sudo),
29(audio), 30(dip), 44(video), 46(plugdev),
105(scanner), 111(netdev), 112(lpadmin), 115(fuse)
```

Changing Password

- passwd changes the password

```
kirk@enterprise:~$ passwd
Changing password for kirk.
(current) UNIX password:
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
```

Switching User

- su *newuser* switches the current user by launching a new shell as *newuser*

Another user on the system

```
kirk@enterprise:~$ su - spock
Password:
spock@enterprise:~$ pwd
/home/spock
spock@enterprise:~$ ls
counsil  pon_farr
spock@enterprise:~$ exit
kirk@enterprise:~$
```

A new shell is spawned

Spock's files

Back to Kirk's shell

Change File Permissions

- chmod *[ugoa][+-][rwx] file* changes the permissions of *file*
- *u*, *g*, *o* and *a* stand of user, group, others and all
- + and - stand for add or remove operation
- *r*, *w* and *x* stand for read, write and execute permissions

```
kirk@enterprise:~# ls -l
-rw-r--r-- 1 kirk bridge 22 Jun 18 01:13 script
```

Original permissions

“all”, “add”, “execute”

```
root@enterprise:~# chmod a+x script
root@enterprise:~# ls -l
-rwxr-xr-x 1 spock science 22 Jun 18 01:13 script
```

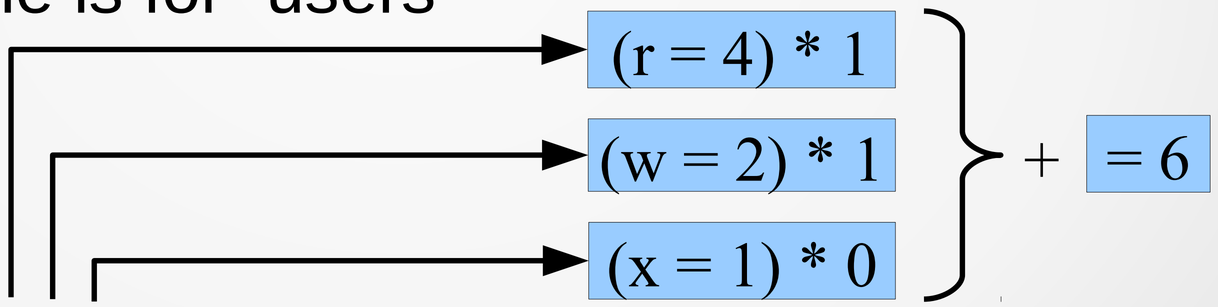
New permissions: added execute permission to all

Change Permissions: Examples

- o-x means remove executable permission to others
- g+w means grant write permission to group
- a-r means remove read permission from all

Change File Permissions (contd.)

- chmod *octalperms file* changes all the permissions of a *file*
- Permissions can be represented as numers in octal base
- r equals 4, w equals 2, x equals 1
- Least significant digit is for “others”, next one is for “group” and the next one is for “users”
- Examples:



  - 644 means `rw-r--r--`
  - 755 means `rwxr-xr-x`
  - 400 means `r--------`

Change Ownership of a File

- *chown user.group filename* changes the user owning the file *filename* to *user* and group owning the file to *group*
- Need to be root to change ownership
- Non-root user can change group ownership if belonging to the target group

```
root@enterprise:~# ls -l
-rw-r--r-- 1 kirk bridge 22 Jun 18 01:13 script
```

Original user and group ownership

```
root@enterprise:~# chown spock.science script
root@enterprise:~# ls -l
-rw-r--r-- 1 spock science 22 Jun 18 01:13 script
```

New user and group ownership

Remote Login

Open Shell on a Remote Machine

- ssh *user@hostname* opens a shell on the remote machine

```
kirk@ent:~$ ssh spock@192.168.36.1
spock@192.168.36.1's password:

spock@vulcan:~$ ls
counsil pon_farr

spock@vulcan:~$ exit
logout
Connection to 192.168.36.1 closed.
kirk@ent:~$
```

user

hostname

Prompts for password

Command executed on remote shell

Back to Kirk's shell

Executing Commands on a Remote Machine

- ssh *user@hostname command* runs command on the remote machine

```
kirk@ent:~$ ssh spock@localhost ls
counsil  pon_farr

kirk@ent:~$ ssh spock@localhost cat counsil
Vulcan science counsil
```

$$\underbrace{\texttt{cat counsil}}_{\text{command}}$$

List of Logged in Users

- who to see the list of users logged in

```
kirk@enterprise:~$ who
mccoy    pts/1    2014-08-10 21:58 (:0)
kirk     pts/2    2014-08-10 22:03 (bridge)
scott    pts/3    2014-08-10 13:04 (eng)
spock    pts/4    2014-08-09 02:07 (vulcan)
```

Login name

Terminal

Login time

Remote hostname or graphical display

Announcements

- wall announces a message to all users

```
kirk@enterprise:~$ wall
This is the captain.
All hands, RED ALERT!
{Control-D}
```

To all logged in users

```
scott@enterprise:~$
Broadcast Message from kirk@enterprise
        (/dev/pts/1) at 22:04 …
This is the captain.
All hands, RED ALERT!
```

Some More Interesting Commands

File Manipulation

- find to recursively find files matching a complex criteria
- xargs to convert input into arguments
- tar, zip, bzip, xz for archiving and compression
- locate to find files using an indexed database
- alias for setting an alias to an existing command

Text Processing

- file to know the type of a file looking at the contents
- sed for performing text translations
- awk a programming language for scanning and processing patterns
- diff to show differences between two files

System Administration

- du to find the size occupied by file on disk
- df for seeing free disk space
- free to see the memory usage
- top to monitor running processes
- shutdown, reboot to restart machine
- crontab for scheduling tasks
- mount, umount for accessing other disks/partitions
- yum, apt-get for software package installation/removal
- wget to download a file from a website

References

- Man Pages: *man*
- Info Documentation: *info*
- Learning the Shell: http://linuxcommand.org/learning_the_shell.php