

CST8177 – Linux II

Review of Fundamentals (cont'd)

Topics

- ▶ change your password on CLS if you haven't already
- ▶ the filesystem
- ▶ access permissions
- ▶ symbolic links
- ▶ hard links

Variables

- ▶ Variables for general use (variables that are not environment variables) have lower case names
- ▶ Environment variables are indicated by their UPPER CASE names: SHELL, VISUAL, etc
- ▶ It's usually best to put variable expansions inside double quotes, to protect any special characters that might be inside the variable:

echo "\$somevar"

- if somevar contained the * character, the double quotes stop the shell from globbing it

Setting Variables

- ▶ set the variable `myvar` to have value `value`

```
myvar=value
```

- ▶ Note, to make this variable setting visible in sub processes we use `export`

```
export myvar=value
```

or

```
myvar=value
```

```
export myvar
```

Variable setting for command

- set the `myvar` variable to have a null value, then run the `value` command with that variable setting in effect

```
myvar= command
```

- ▶ Notice that if you try mistakenly use this to try to set the value of `myvar` to `value`

```
myvar= value
```

in this case you are actually trying to run a command called `value`

Variable setting for command (cont'd)

The usual way to use this mechanism is something like

```
VISUAL=nano vipw
```

- ▶ This means to set the value of the environment `VISUAL` variable to `nano`, and use that while the `vipw` command runs

Setting Variables Mistakes

- set the `myvar` variable to have a null value, then run the `value` command with that variable setting in effect

```
myvar= value
```

- run the `myvar` command with one argument, namely `=value`

```
myvar =value
```

- ▶ run the `myvar` command with two arguments, namely `=` and `value`

```
myvar = value
```

Unix Files

- ▶ Sobel, Chapter 6
- ▶ [160_pathnames.html](#) *Unix/Linux Pathnames (absolute, relative, dot, dot dot)*
- ▶ [450_file_system.html](#) *Unix/Linux File System – (correct explanation)*
- ▶ [460_links_and_inodes.html](#) *Hard links and Unix file system nodes (inodes)*
- ▶ [460_symbolic_links.html](#) *Symbolic Links – Soft Links – Symlinks*
- ▶ [500_permissions.html](#) *Unix Modes and Permissions*
- ▶ [510_umask.html](#) *Umask and Permissions*

File Permissions

```
cst8177.idallen.ca - PuTTY
[wen99999@idallen-ubuntu ~]$ ls -ail
total 76
5242914 drwxr-x---  6 wen99999 wen99999  4096 Sep 10 20:14 ./
          2 drwxr-xr-x 675 root      root      20480 Sep 12 11:56 ../
5242915 -rw-----  1 wen99999 wen99999  9804 Sep 12 19:45 .bash_history
5242946 -rw-rw-r--  1 wen99999 wen99999   17 Sep  9 13:43 .bash_profile
5242948 -rwxr---x  1 wen99999 wen99999  1079 Sep  9 14:05 .bashrc
5242942 -rw-rw-r--  1 wen99999 wen99999  1061 Sep  9 14:02 .bashrcbak
5242937 drwxrwxr-x  2 wen99999 wen99999  4096 Sep  9 15:31 bin/
5242916 drwx-----  2 wen99999 wen99999  4096 Sep  9 10:45 .cache/
5242918 drwxr-xr-x  3 wen99999 wen99999  4096 Sep  9 10:50 CST8177-14F/
5242929 drwxrwxr-x  3 wen99999 wen99999  4096 Sep  9 11:36 CST8177-14W/
5242956 -rwxrw-r--  1 wen99999 wen99999   112 Sep  9 15:39 ls
5242949 -rw-----  1 wen99999 wen99999  7922 Sep 10 20:14 .viminfo
[wen99999@idallen-ubuntu ~]$ █
```

Typical directory and file

inode 5242914
drwxr-x---
access time
modification time
change time
...etc...

.	inode 5242914
..	
.bash_history	inode 5242915
.bash_profile	inode 5242946
...etc...	...etc...

inode 5242946 -
rw-rw-r-- access
time
modification time
change time
...etc...

data blocks for
the file
there is no
filename here
the filename(s)
(at least one) are
stored in
directories

File Permissions (cont'd)

inode 5242914

drwxr-x---

access time

modification time

change time

...etc...

Need read (r) on directory to read this column

.	inode 5242914
..	
.bash_history	inode 5242915
.bash_profile	inode 5242946
...etc...	...etc...

Need search (x) on directory to access this column

Need write (w) *and* search (x) on directory to change first column

File Permissions (cont'd)

inode 5242946
-rw-rw-r--
access time
modification time
change time
...etc...

Need search (x) on
directory this file is in
to access this info on
the file's inode

data blocks for
the file
there is no
filename here
the filename(s)
(at least one) are
stored in
directories

Need read (r) / write
(w) / execute (x) on **file**
to read / write / execute
this file (contents)

File Attributes

Field No.	Stat Name	Unix	Win98/NT	MacOS
1	st_dev	Device number of filesystem	Drive number	vRefNum
2	st_ino	Inode number	Always 0	fileID/dirID
3	st_mode	File mode	File mode	777 dirs/apps; 666 docs; 444 locked docs
4	st_nlink	Number of links to the file	Number of link (only on NTFS)	Always 1
5	st_uid	Owner ID	Always 0	Always 0
6	st_gid	Group ID	Always 0	Always 0
7	st_rdev	Device ID for special files	Drive No.	Always 0
8	st_size	File size in bytes	File size in bytes	Data fork file size in bytes
9	st_blksize	Preferred block size	Always 0	Preferred block size
10	st_blocks	Number of blocks allocated	Always 0	Number of blocks allocated
11	st_atime	Last access time since epoch	Last access time since epoch	Last access time -66 years
12	st_mtime	Last modify time since epoch	Last modify time since epoch	Last access time -66 years
13	st_ctime	Inode change time since epoch	File create time since epoch	File create time -66 years

Information given by long listing: ls -l

10 characters

- file type as the first letter
- access modes (remaining letters)

Link count

- number of links to this file or directory

User-owner Login Name

- user who owns the file/directory
- based on owner UID

User-owner Group Name

- group who owns the file/directory
- based on owner GID

File Size

- size (in bytes or K) of the file/directory

Date/Time Modified

- date and time when last created / modified / saved

File Name

- actual file/directory name

File Types

- ▶ Linux recognizes and identifies several file types, which is coded into the first letter of the first field of information about the file:
 - ▶ **-** (**dash**) a regular file
 - ▶ **b** block device special file
 - ▶ **c** character device special file
 - ▶ **d** a directory
 - ▶ **l** a symbolic (soft) link
 - ▶ **p** a named pipe or FIFO
 - ▶ **s** socket special filename

File Access Privileges

- In Linux, 3 types of access permissions or privileges can be associated with a file:
 - **read (r)** grants rights to read a file
 - **write (w)** grants rights to write to, or change, a file
 - **execute (x)** grants rights to execute the file (to run the file as a command)
- All 3 permissions can then be applied to each of 3 types of users:
 - **User:** owner of the file
 - **Group:** group to which user must belong to gain associated rights
 - **Others:** not **User** and not member of **Group** (sometimes called “World” or “Everybody”)

Octal representation of permissions

<i>Octal</i>				
<i>r</i>	<i>w</i>	<i>x</i>	<i>Value</i>	<i>Meaning</i>
0	0	0	0	No permission
0	0	1	1	Execute-only permission
0	1	0	2	Write-only permission
0	1	1	3	Write and execute permissions
1	0	0	4	Read-only permission
1	0	1	5	Read and execute permissions
1	1	0	6	Read and write permissions
1	1	1	7	Read, write and execute permissions

Directory Access Privileges

- ▶ The same three types of access permissions or privileges are associated with a directory, but with some differences:
 - **read (r)** rights to read the directory
 - **write (w)** rights to create or remove in the directory
 - **execute/search (x)** rights to access the directory meaning, cd into the directory, or access inodes it contains, or “pass through”

All three permissions can then be applied to each of three types of users as before.

- **User** owner/creator of the file
- **Group** group to which user must belong
- **Others** everyone else (Rest-of-world)

Linux File Permissions

- ▶ Three special access bits. These can be combined as needed.
- ▶ SUID - Set User ID bit
 - When this bit is set on a file, the effective User ID of a process resulting from executing the file is that of the owner of the file, rather than the user that executed the file
 - For example, check the long listing of /usr/bin/passwd – the SUID bit makes this program run as root even when invoked by a regular user – allowing regular users to change their own password

chmod 4xxx file-list

chmod u+s file-list

Linux File Permissions

- ▶ SGID - Set Group ID bit
 - Similar to SUID, except an executable file with this bit set will run with effective Group ID of the owner of the file instead of the user who executed the file.

chmod 2xxx file-list

chmod g+s file-list

Linux File Permissions

- ▶ sticky bit (restricted deletion flag)
 - The sticky bit on a directory prevents unprivileged users from removing or renaming a file in the directory unless they are the owner of the file or the directory
 - for example, **/tmp** is a world-writeable directory where all users need to create files, but only the owner of a file should be able to delete it.
 - without the sticky bit, hostile users could remove all files in /tmp; whereas with the sticky bit, they can remove only their own files.

chmod 1xxx dir-list

chmod +t dir-list

Linux File Permissions

- ▶ The permissions a user will have is determined in this way:
 - If the user is the owner of the file or directory, then the user rights are used.
 - If the user is not the owner but is a member of the group owning the file or directory, then the group rights are used.
 - If the user is neither the owner nor a part of the group owning the file, then the other rights are used.
- ▶ NOTE: It is possible to give the “world” more permissions than the owner of the file. For example, the unusual permissions `-r--rw-rw-` would prevent only the owner from changing the file – all others could change it!

umask

- The permissions assigned to newly created files or directories are determined by the **umask** value of your shell.
- Commands:
 - **umask** - display current umask
 - **umask xyz** - sets new umask to an octal value **xyz**
- permissions on a newly created file or directory are calculated as follows:
 - start with a “default” of 777 for a directory or 666 for a file
 - for any 1 in the binary representation of the umask, change the corresponding bit to 0 in the binary representation of the default
 - umask is a reverse mask: the binary representation tells you what bits in the 777 or 666 default will be 0 in the permissions of the newly created file or directory

umask examples (Files)

▶ if umask is 022

- binary umask representation: $000010010 = 022$
- default file permissions 666: 110110110
- permissions on new file: $110100100 = 644$

▶ if umask is 002

- binary umask representation: $000000010 = 002$
- default file permissions 666: 110110110
- permissions on new file: $110110100 = 664$

▶ if umask is 003

- binary umask representation: $000000011 = 003$
- default file permissions 666: 110110110
- permissions on new file: $110110100 = 664$

umask examples (Files, cont'd)

- ▶ notice that for files, a umask of 003 ends up doing the same thing as a umask of 002
- ▶ Why?

umask examples (Directories)

▶ if umask is 022

- binary umask representation: $000010010 = 022$
- default dir permissions 777: 111111111
- permissions on new dir : $111101101 = 755$

▶ if umask is 002

- binary umask representation: $000000010 = 002$
- default dir permissions 777: 111111111
- permissions on new dir : $111111101 = 775$

▶ if umask is 003

- binary umask representation: $000000011 = 003$
- default dir permissions 777: 111111111
- permissions on new dir : $111111100 = 774$

umask examples (Dirs, cont'd)

- ▶ notice that for directories, a umask of 003 gives different results than a umask of 002
- ▶ Why?

Linux File Permissions

- It is important for the Linux file system manager to govern permissions and other file attributes for each file and directory, including
 - ownership of files and directories
 - access rights on files and directories
 - The 3 timestamps seen in **stat (man stat)**
- The information is maintained within the file system information (**inodes**) on the hard disk
- This information affects every file system action.

Linux Basic Admin Tools

▶ **chown owner[:group] files**

- Change ownership of files and directories (available for **root** only)

Examples:

chown guest:guest file1 dir2

- change ownership of **file1** and **dir2** to user **guest** and group **guest**

chown guest dir2

- change ownership of **dir2** to user **guest** but leave the group the same

chown :guest file1

- change ownership of **file1** to group **guest** but leave the user the same (can also use **chgrp**)

Linux Basic Admin Tools

▶ **chmod permissions files**

- Explicitly change file access permissions

Examples:

chmod +x file1

- changes **file1** to have executable rights for user/group/other, subject to umask

chmod u+r,g-w,o-rw file2

- changes **file2** to add read rights for user, remove write rights for group and remove both read and write rights for others

chmod 550 dir2

- changes **dir2** to have only read and execute rights for user and group but no rights for other

Extending Unix

- ▶ create a command with basic scripting
 - put “#!/bin/sh -u” at very beginning of file
 - **PATH=/bin:/usr/bin ; export PATH**
 - **umask 022**
 - put commands in file
 - make file executable
- ▶ put the file in a directory that is in \$PATH
- ▶ http://teaching.idallen.ca/cst8207/14w/notes/400_search_path.html
- ▶ Not a good idea to put “.” in PATH
- ▶ Security implications of putting “current directory” , “.” in PATH
- ▶ `PATH=.:$PATH`
- ▶ demonstration of how the bad guy can arrange for you to inadvertently run their malicious commands as you