CSNB113: System Administration
-
5th Topic: Filesystem Layout // Use and Examples

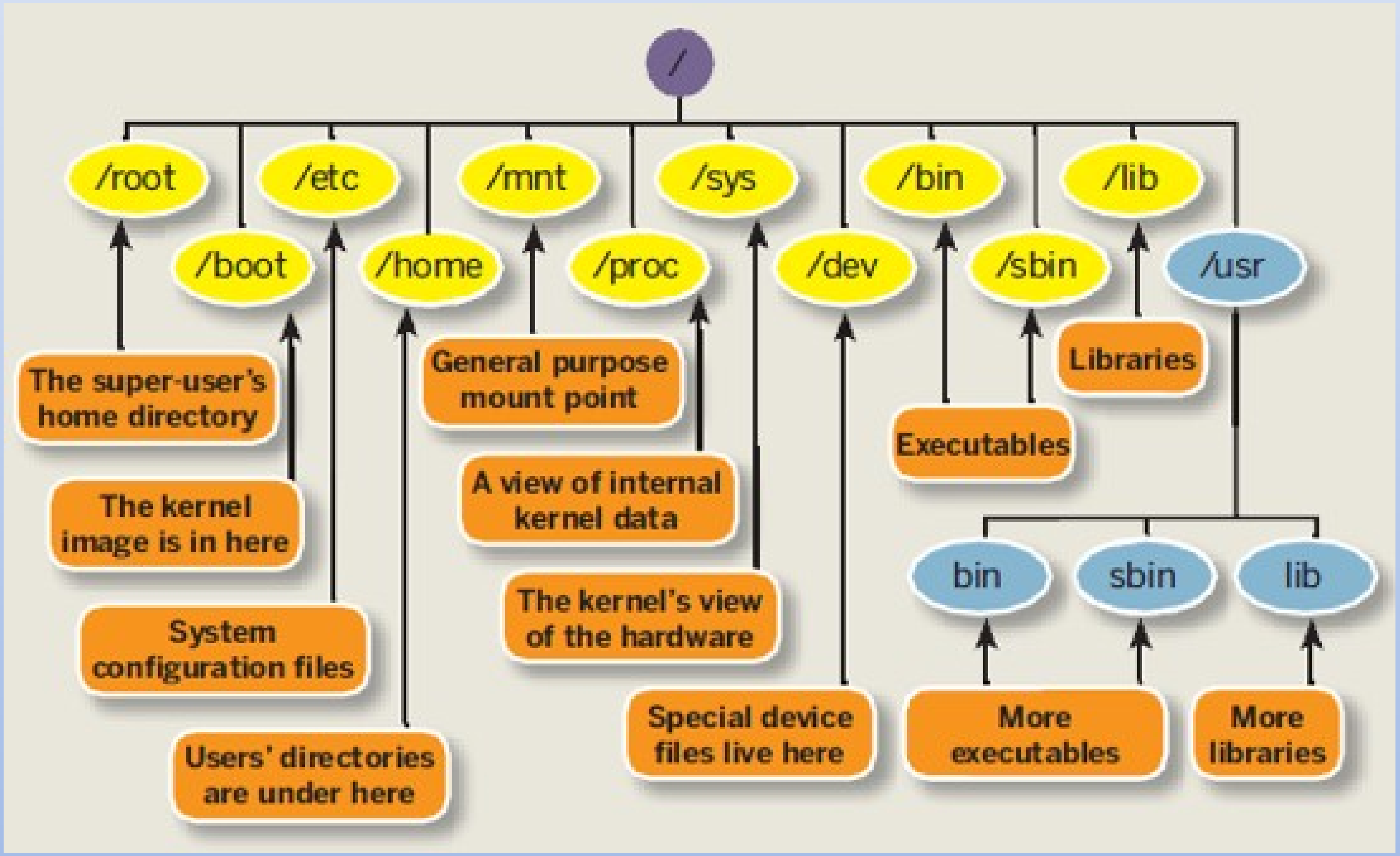
File System Layout

The file system on *nix is rather complex / complicated. Fortunately, the users do not need to know.

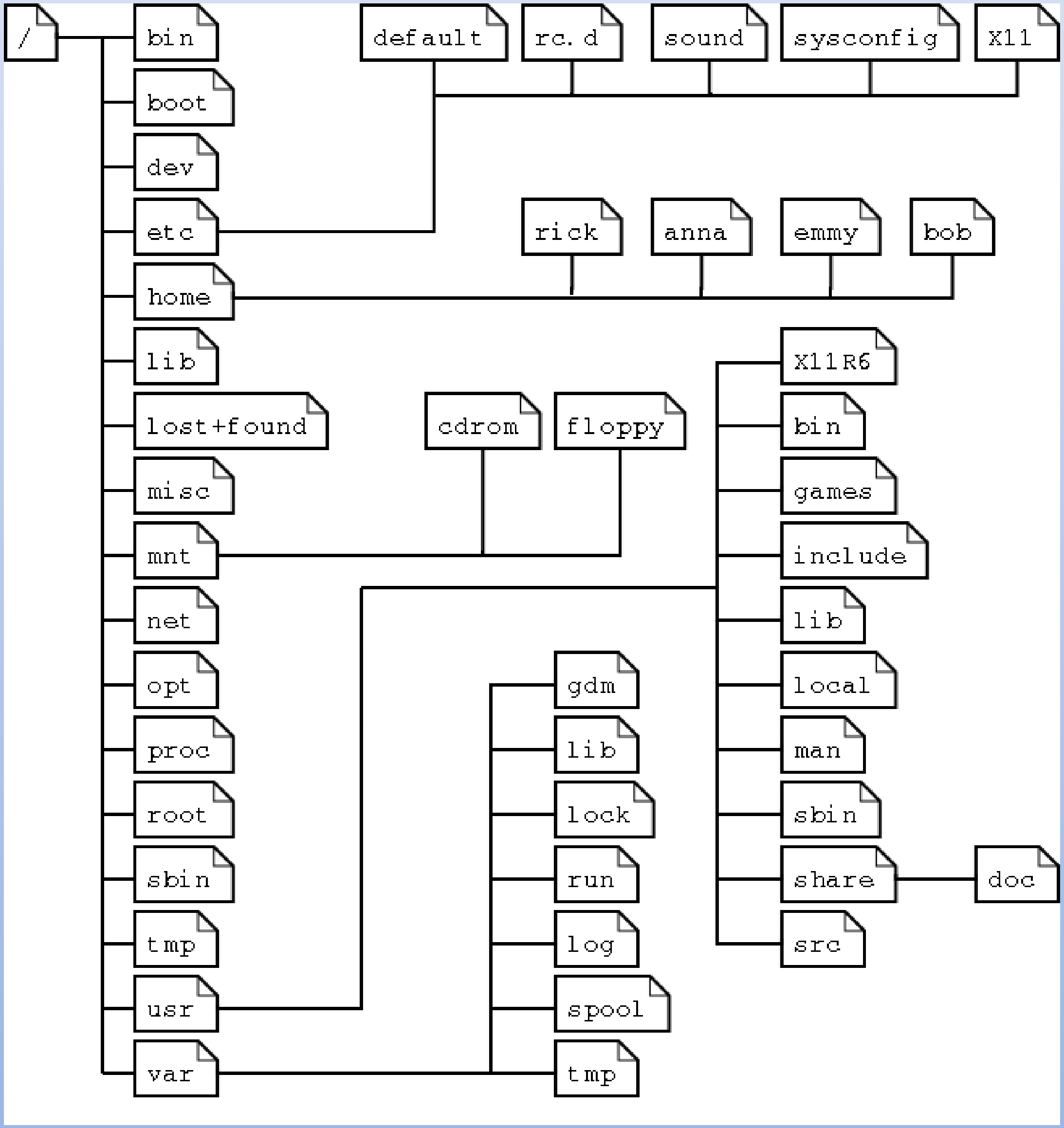
The advantage for the system administrator is, that there are defined locations for almost each and everything.

- /home: all user data
- /etc: all configuration data
- /root: all data accessible for root, the system administrator, only
- /proc: information about the running processes, and the kernel (Linux only)
- /dev: location for all devices, drives. Example: mouse, keyboard, graphics adapter, hard disk, thumb drives, consoles, etc.
- /tmp: all rapidly changing information – none that is relevant for running the system. It is rather a scrap-board or a temporary storage for the system.

File System Layout - simple


/
/root
/boot
/etc
/home
/mnt
/proc
/sys
/dev
/bin
/sbin
/lib
/usr
The super-user's home directory
The kernel image is in here
System configuration files
Users' directories are under here
General purpose mount point
A view of internal kernel data
The kernel's view of the hardware
Special device files live here
Executables
Libraries
bin
sbin
lib
More executables
More libraries

File System Layout - detail



Navigation - I

The navigation on these directories is done using the change directory command ('cd').

cd is very versatile:

cd	alone sends each user back to his/her home directory
cd /	sends the user to the root-directory of the filesystem
cd ..	changes to the upper directory:

```
$ pwd
/home/users/udippel
$ cd ..
$ pwd
/home/users
$ cd ..
$ pwd
/home
$ cd
$ pwd
/home/users/udippel
$ cd /
$ pwd
/
$
```

Navigation - II

cd is very versatile:

It is possible to 'climb' one or more up, and then one or more down:

```
$ pwd
/home/users/udippel
$ cd ../../ftp/pub/ubuntu/
$ pwd
/home/ftp/pub/ubuntu
$
```

The term here is **relative path**, because you start from where you happen to be, and ../ gets you into a directory higher.

```
$ pwd
/home/users/udippel
$ cd /home/ftp/pub/ubuntu/
$ pwd
/home/ftp/pub/ubuntu
$
```

This here is using the **abolute path**, because you start 'absolutely', from '/'; due to the first slash.

Navigation - II

cd is very versatile:

It is possible to 'climb' one or more up, and then one or more down:

```
$ pwd
/home/users/udippel
$ cd ../../ftp/pub/ubuntu/
$ pwd
/home/ftp/pub/ubuntu
$
```

The term here is **relative path**, because you start from where you happen to be, and ../ gets you into a directory higher.

```
$ pwd
/home/users/udippel
$ cd /home/ftp/pub/ubuntu/
$ pwd
/home/ftp/pub/ubuntu
$
```

This here is using the **abolute path**, because you start 'absolutely', from '/'; due to the first slash.

Navigation - III

To test the understanding:

```
$ pwd
/home/users/udippel
$ cd /etc
$ pwd
[???]

$ pwd
/home/users/udippel
$ cd etc
$ pwd
[???]
```

... and if the directory does not yet exist? Then we have to create it:

mkdir [**m**a**k**e **dir**ectory]

And how can we search or find a file?

Find - Locate

The command 'find' is very helpful, and allows all sorts of options: searching files according to name, date, permissions, modification time, last access time, if it is a regular file or a directory, etc.

The disadvantage is the **syntax**, which is complex for all these. Also, when you have a lot of files, it takes a long time, because all files need to be accessed (similar to the 'Search' on Windows).

Another method is 'locate'. This is a search in a database for the name or a part of the name. This is **very fast**, but it only searches the database. A file created more recently will not show. So you need to update the database (command: 'updatedb').

We start searching a file that does NOT exist on the system: find_me_not

```
$ find -name find_me_not [file is not found – to be expected]
$ locate find_me_not  [file is not found – to be expected]
$ touch find_me_not   [touch creates an empty file here]
$ find -name find_me_not
./find_me_not [file is found – to be expected]
$ locate find_me_not  [file is not found – to be expected as the database has not been updated]
$ sudo updatedb   [updating the database of files and directories]
$ locate find_me_not
/home/udippel/find_me_not          [file is found – to be expected now, after the database update]
$
```

Wait for the lab exercise or try out yourself, how much faster locate is.

Permissions - Changing

The list of files and directories comes with ls, or ls -l for the long list, with many details.
We can see the permissions; we can see who has which rights on a file.
We have almost all tools needed for basic system administration. We still need to learn how to change these items.

```
-|rwx|r-x|r-x| 1 root daemon 22936 2010-06-11 15:24 /usr/bin/whoami
-|421|4-1|4-1|
-| 7 | 5 | 5 |
```

The **permission**(s) of this file are 755.
Wanting to give these permissions to a file, the command is:

```
$ chmod 755 /usr/bin/whoami
```

The owner of the file is root, it belongs to group deamon.
To change ownership:

```
$ chown fred /usr/bin/whoami
```

changes the ownership to user fred

To change the group:

```
$ chgrp student /usr/bin/whoami
```

makes the file belong to group student

Removing, Deleting

The easier part is getting rid of files and directories:

```
$ rm /usr/bin/whoami
```

removes the file /usr/bin/whoami in directory /usr/bin/, or any other file of your liking – as long as you have the 'w' permission for it!

```
$ rmdir /usr/bin/
```

in turn **rem**oves **dir**ectory /usr/bin/. There is not much of danger using this command, because it will not remove anything, as long as there is at least one file in that directory. rmdir only removes totally empty directories.

These commands are 'boring', because in order to delete a directory 'demo' containing 21 files, one needs to delete these 21 files, then cd .., followed by rmdir demo.

This works, but is too cumbersome.

Globbing

Globbing is the way to 'summarise' a number of files according to their names. This is being done by ***expanding* wildcards**.

Task	**Example**	**Unix**	**Windows Powershell**
Match one character	?at matches cat, bat, mat, Fat	?	?
Match any number of characters	Uni* matches Unity, Uniten, Uni – but not unity	*	*
Match specific characters	[d,F] matches demo, Fish – but not Demo, fish, cat	[chars]	[chars]
Escape	Uniten\? matches Uniten? only	\	`

Globbing - Examples

```
$ ls -l
 -rw-rw-r--    1 bozo  bozo         0 Aug  6 18:42 a.1
 -rw-rw-r--    1 bozo  bozo         0 Aug  6 18:42 b.1
 -rw-rw-r--    1 bozo  bozo         0 Aug  6 18:42 c.1
 -rw-rw-r--    1 bozo  bozo       466 Aug  6 17:48 t2.sh
 -rw-rw-r--    1 bozo  bozo       758 Jul 30 09:02 test1.txt

$ ls -l t?.sh
 -rw-rw-r--    1 bozo  bozo       466 Aug  6 17:48 t2.sh

$ ls -l [ab]*
 -rw-rw-r--    1 bozo  bozo         0 Aug  6 18:42 a.1
 -rw-rw-r--    1 bozo  bozo         0 Aug  6 18:42 b.1

$ ls -l [a-c]*
 -rw-rw-r--    1 bozo  bozo         0 Aug  6 18:42 a.1
 -rw-rw-r--    1 bozo  bozo         0 Aug  6 18:42 b.1
 -rw-rw-r--    1 bozo  bozo         0 Aug  6 18:42 c.1

$ ls -l *
 -rw-rw-r--    1 bozo  bozo         0 Aug  6 18:42 a.1
 -rw-rw-r--    1 bozo  bozo         0 Aug  6 18:42 b.1
 -rw-rw-r--    1 bozo  bozo         0 Aug  6 18:42 c.1
 -rw-rw-r--    1 bozo  bozo       466 Aug  6 17:48 t2.sh
 -rw-rw-r--    1 bozo  bozo       758 Jul 30 09:02 test1.txt
```

Removing of bulk data

To remove all files from the directory /usr/bin:

```
$ rm /usr/bin/*
```

re**m**oves all files (* matches all files) in /usr/bin/

Then,

```
$ rmdir /usr/bin/
```

in turn **rem**oves **dir**ectory /usr/bin/.

This does not work for

```
$ rm /usr/*
```

Why? This command removes all **files** in /usr/; but **not** the **directories**, like /usr/bin/. So you would have to **recurse** into all the **subdirectories** under /usr/ and first delete all the files (see above, for /usr/bin), and then the directories.

This is still cumbersome, therefore we have an option for **recursion**:

Removing of ***all*** data

```
$ rm -R /usr/*
```

This will recurse into all sub-directories under /usr/, and the sub-sub-directories ..., delete all files there, then delete the subdirectories, finally delete all files in /usr/, and at last delete the directory /usr/ itself.
If this command finds reasons to *double-check*, and asks you questions, you have to type 'y'. Boring? Option -f ('force') stops 'quarreling' with you:

```
$ rm -Rf /usr/*
```

With this command, all and anything under /usr/ is gone, **irrevocably**.

In case you happen to be in '/' the root directory, though you think you are in some subdirectory,

```
$ rm -Rf *
```

Simply removes all and everything you have on your drives, from all partitions. 100%. Success guaranteed. Happy new install. Hopefully, you had made a backup for your data!?

References

- http://en.wikipedia.org/wiki/Glob_(programming)
- http://www.faqs.org/docs/abs/HTML/globbingref.html
- Textbook, p.697 – 702 "Globbing With Wildcards"
- Textbook, p.328, "WHAT'S IN A NAME?"
-