Files

A normal "flat" file is a collection of information. It's usually stored somewhere reasonably non-volatile — when you turn off your power, it's nice to still have your files around when you power back up.

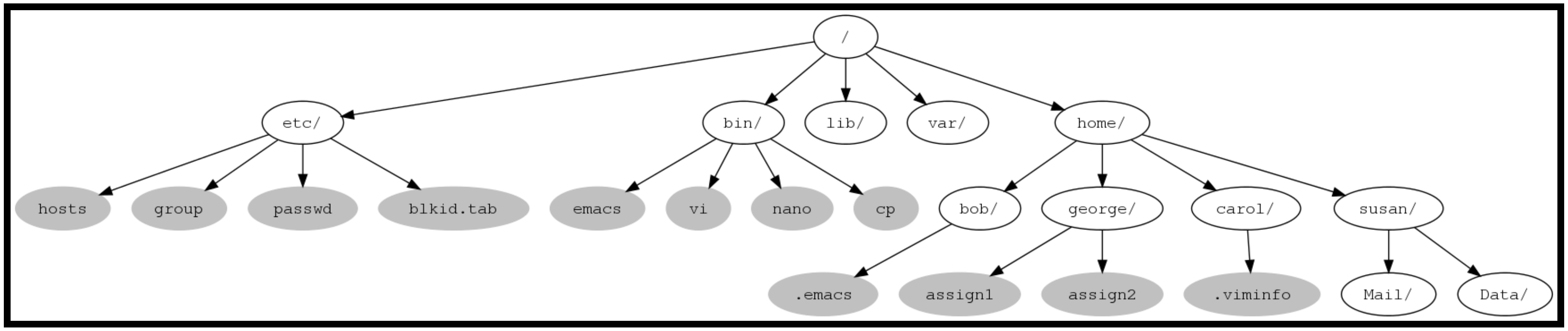
You can reasonably view a Unix file as a stream of bytes; as Kernighan and Pike put it in the The Unix Programming Environment, "A file is a sequence of bytes. ... No structure is imposed on a file by the system, and no meaning is attached to its contents — the meaning of the bytes depends solely on the programs that interpret the file. ... Magnetic tapes, mail messages, characters typed on the keyboard, line printer output, data flowing in pipes — each of these files is just a sequence of bytes as far as the system and the programs in it are concerned." (page 41)

Although this seems quite normal to us these days, back in the early days of Unix, such a simple view of files was not the norm; instead, many files were structured in some manner, generally by some sort of "record" scheme, though there were many other ideas floating around.

So what are some of the common ways that we interpret the contents of a file? The simplest of course for human being is simply as a text file. The simplest for a computer might be as a binary program, where its contents are loaded into memory and directly executed by a processor. Other types include PDF files, spreadsheets, SVG files, database files, and a host of other types.

But isn't there some way to indicate what the purpose of a file is? Some operating systems, for instance, have used the idea of indicating types in the file names: "letter.pdf", for instance, in such a scheme, would be a PDF file. But Unix doesn't use that sort of scheme. Instead, it has the idea of a "magic number" at the beginning of a file to indicate the file type.

We aggregate these files into what we call "directories". Directories are recursive structures, since they can also contain other directories.


/
etc/
bin/
lib/
var/
home/
hosts
group
passwd
blkid.tab
emacs
vi
nano
cp
bob/
george/
carol/
susan/
.emacs
assign1
assign2
.viminfo
Mail/
Data/

Pathnames

We need to be able to name the locations of our files.

The "root" is the starting point, referred to with "/"

Sub-directories identified by name, separated by "/"

Absolute pathnames always start with the "root" (i.e., the slash character)

Relative pathnames are from current directory

Going up the directory tree

".." refers to directory above

"." refers to the current working directory

Referencing user directories:

`~USER` represents the absolute path to `USER`'s home directory

`"~/"` represents the absolute path to your own home directory

Directory Structure

- The leading item is called the "root", written simply as "/"
- Items in a subgraph are "contained" by the directories in the path leading back to the root

Pathnames

Absolute pathnames simply give directions on how to get to a node

Examples of absolute pathnames:

`"/bin/emacs"`

`"/home/carol/Mail/"`

Relative pathnames are relative to the current working directory

Examples of relative pathnames

`"Mail/"`

`"../../bin/emacs"`

Commands

When you type an initial element at a shell prompt, you might be doing one of three things:

- Starting some sort of aggregation, such as a loop or an "if/then" structure
- Invoking a "built-in", such as `cd`
- Calling a program, such as `emacs`

Typical command structure:

```
COMMANDNAME [FLAGS] [PARAMETERS]

FLAGS: Commands often accept one or more flags after command name.
       Each flag starts with "-" or with "--", and is separated
       from other flags by spaces.
       Individual flags often may be combined with a single "-".

ls -l -a
ls -la

PARAMETERS: Commands often accept one or more parameters.
            Parameters are often pathnames representing files
            or directories.
            Parameters are separated by spaces.

cp -a /home/carol/Mail /home/carol/Mail-Backup
```

Finding commands

Generally, you don't want to type the entire pathname for a program like Perl: `/usr/bin/perl`

Instead, it would be nice to use the "relative" syntax of simply `perl` — but you don't want to have to also first `cd` to `/usr/bin` either.

```
PATHH
     HPATHH              H          H
      HT  HT     H          H  A  H  A
                      H     A         A
```

```
$ echo $PATH
/usr/kerberos/bin:/usr/local/bin:/bin:/usr/bin
```

This `PATH` means that each of the paths `/usr/kerberos/bin`, `/usr/local/bin`, `/bin`, and `/usr/bin` will be searched when an unadorned command name is invoked.

Operating on files

Command	Purpose
ls	List files and directories
cat	Output a bytestream
less	Paginate a bytestream
pr	Paginate a bytestream
touch	Update the timestamp on a file; if the file does not exist, it is created
cd	Change the shell's current working directory (built-in)

Command	Purpose
cp	Copy files and directories
mv	Move files and directories
rm	Remove files and directories
rmdir	Remove empty directories
mkdir	Create directories

Command	Purpose
wc	Count lines, words, and characters in a file
grep	Search a file
sort	Sort a file
head	Show the initial lines of a file
tail	Show the terminal lines of a file
cmp	Find the first difference in two files
diff	Find the differences in two files

Three very useful additional commands

Command	Purpose
man	Read the fine manual
date	Find date and time
who	List the people logged in

Permissions

Every file and directory has a set of permissions associated with it.

```
$ ls -l /etc/hosts
-rw-r--r-- 1 root root 905 Jun 16  2010 /etc/hosts
$ ls -dl /etc
drwxr-xr-x 105 root root 12288 Jan 15 14:24 /etc
$ ls -l /usr/bin/emacs-21.4-x
-rwxr-xr-x 2 root root 6649776 Apr 28  2011 /usr/bin/emacs-21.4-x
$ ls -l /usr/bin/emacs
lrwxrwxrwx 1 root root 19 Feb  8  2012 /usr/bin/emacs -> /usr/bin/emacs-21.4
```

The first component tells us about the item in the filesystem; an unadorned "-" indicates a regular file; "d" indicates a directory; "l" indicates a soft link; "s" indicates a socket; "c" is a character device file; "b" is a block device file; "p" is a named pipe.

```
$ ls -l /etc/hosts
-rw-r--r-- 1 root root 905 Jun 16  2010 /etc/hosts
$ ls -dl /etc
drwxr-xr-x 105 root root 12288 Jan 15 14:24 /etc
$ ls -l /usr/bin/emacs-21.4-x
-rwxr-xr-x 2 root root 6649776 Apr 28  2011 /usr/bin/emacs-21.4-x
$ ls -l /usr/bin/emacs
lrwxrwxrwx 1 root root 19 Feb  8  2012 /usr/bin/emacs -> /usr/bin/emacs-21.4
```

The next 3 characters indicate whether the owner of a file has (1) read (2) write or (3) execute permission (execute in the case of a directory means being able to cd into the directory). The second set of three characters indicate whether users in the same group as this item have read/write/execute permission; the last group of three indicates if all users (the "world") have read/write/execute permission.

Command Reference

```
ls [-a][-l][-p][-r][-R][-x] [pathname]

Description: Lists the files in a directory.

Options:
 [-a] Display all files
 [-l] Displays all information
 [-r] Reverses order
 [-R] Includes sub-directories

Examples:
  ls
  ls -al
  ls -al *.exe
```

```
touch filename

Description: Immediately creates an empty file, or updates the
             time stamp on an existing file.
```

```
cp [-i][-R][-r] source destination

Description: Copies the contents of a file or directory to another
             file or directory.

Options:
 [-a] Archive mode; preserve ownership, permissions, and any special attributes
 [-i] Ask before you replace
 [-r] Recursive copy

Parameters:
 source - What you want to copy
 target - New location

Example:
 cp .plan .plan.backup
 cp -r ~/public_html/* /tmp
 cp -a /home/carol /home/carol-backup
```

```
mv [-i] source target

Description: Renames or moves a file from one directory to
     another, either with the same name or a different one. Note the
     original file (and name) will no longer exist. This is not a copy.

Options:
  [-i] Prompt you before replacing a file

Parameters:
  source - what you want to copy
  target - where to put it

Examples:
  mv x y
  mv x dir/
  mv -i /etc/passwd-old /etc/passwd
```

```
rm [-i][-r][-f] NAMES

Description: Delete files.

Options:
  [-i] Prompt you before replacing a file
  [-r] Recursive, deletes an entire directory and all contents
       and subdirectories.
  [-f] Forcible removal. Don't give any error messages or ask questions
       even if files don't exist or permissions are awkward.

Parameters:
  NAMES - the names of what to remove


Example:
  rm -rf /
  rm -i this.*
  rm -r /home/carol
```

```
less [filename]

Description: paginates a file or stdin.
```

```
wc [-c][-l][-w] [NAMES]
Description: Counts characters, lines, and/or words in files or stdin.

Options:
 [-c] Number of characters
 [-l] Number of lines
 [-w] Number of words
```

pwd

```
Description: Displays the current working directory.
```

```
cd [directory]

Description: Changes current working directory.

Examples:
  cd /
  cd
  cd /etc
  cd ../public_html/classes
```

```
mkdir DIRECTORIES

Description: Creates new directories.


Examples:
  mkdir classes
  mkdir ../classes/new
```

```
rmdir DIRECTORIES

Description: Removes the specified empty directories. Will not
  remove a directory that has any contents.

Examples:
  rmdir classes/
  rmdir /home/carol/
```

```
man command

Description: Displays the fine manual page for a command.
```