

UNIX for Programmers and Users

“UNIX for Programmers and Users”

Third Edition, Prentice-Hall, GRAHAM GLASS, KING ABLES

Slides adapted from Prof. Andrzej (AJ) Bieszczad

Starting with Unix

- WARNING:
 - There are differences between the textbook and what you will see experience using Ubuntu. The examples in lectures might also be different. Use the online help to get exact syntax and functionality for another Unix system that you might be using. The differences are mechanical, and not conceptual.

Contents

Introduces the following utilities:

**cancel
cat
chgrp
chmod
chown
clear
cp
date
emacs
file
groups**

**head
lp
lpr
lprm
lpq
lpstat
ls
mail
man
mkdir
more**

**mv
newgrp
page
passwd
pwd
rm
rmdir
stty
tail
tset
vi
wc**

Starting with Unix

- **Logging In**
 - In order to use a UNIX system, you must first **log in with a suitable username**.
- **A username is a unique name** that distinguishes you from the other users of the system.
- **Your username and initial password** are assigned to you by **the system administrator**.
- UNIX first asks you for your username by prompting you **with the line “login:”** and then asks for **your password**.

Starting with Unix

- When you enter your password, the letters that you type are not displayed on your terminal for security reasons.
- UNIX is case sensitive, so make sure that the case of letters is matched exactly to those of your password.
- Depending on how your system is set up, you should then see either a \$, a % or another prompt. That is your default shell prompting you to provide some course of action.
- Here's an example login :

UNIX® System V Release 4.0

login : glass

Password : --> What is typed here is secret and doesn't show.

Last login: Sun Feb 15 18:33:26 from dialin

\$ _

Shells

- The \$ or % prompt that you see when you first log in is displayed by a special kind of program called a shell.
- A Shell is a program that acts as a middleman between you and the raw UNIX operating system.
- It lets you run programs, build pipelines of processes, save output to files, and run more than one program at the same time.
- A shell executes all of the commands that you enter.
- The four most popular shells are:
 - the Bourne shell (sh)
 - the Korn shell (ksh)
 - the C shell (csh)
 - the Bash Shell (bash)

Shells

- All of these shells share a similar set of core functionality, together with some specialized properties.
- The Korn shell is a superset of the Bourne shell, and thus users typically choose either the C shell or the Korn shell to work with.
- The Bash shell, or Bourne Again Shell, is an attempt to combine the best features from Korn and C shells.
- Bash is becoming the most popular shell, because it is freely available and comes as a standard shell on Linux systems.

Running a utility

- To run a utility, simply enter its name at the prompt and press the Enter key.
- One utility that every system has is called `date`, which displays the current date and time:

`$ date` --> run the date utility.

Thu, Aug 18, 2016 7:11:59 AM

`$ _`

Running shell

- Running shell

ka 9: `echo $SHELL`

`/usr/bin/csh`

clear

- This utility clears your screen.

man: online help

- All UNIX systems have a utility called **man** (short for **manual page**) that **puts this information** at your fingertips.
- The manual pages are **on-line copies of the original UNIX documentation**. They contain information about **utilities, system calls, file formats, and shells**.

Organization of the manual pages

- The typical division of topics in manual pages ([sections](#)) is as follows:
 1. Commands and Application Programs.
 2. System Calls
 3. Library Functions
 4. Special Files
 5. File Formats
 6. Games
 7. Miscellaneous
 8. System Administration Utilities

Using man

Here's an example of man in action:

```
$ man chmod          ---> select the first manual entry.
CHMOD(1V)            USER  COMMANDS              CHMOD(1V)
NAME
    chmod - change the permissions mode of a file
SYNOPSIS
    chmod C -fR V mode filename ...
                                --> the description of chmod goes here.
SEE ALSO
    csh(1), ls(1V), sh(1), chmod(2V), chown(8)
```

Terminating a Process: Control-c

- There are often times when you **run a program** and then wish **to stop it** before it's finished.
- The standard way to execute this action in UNIX is to press the keyboard sequence **Control-C**.
- Most processes **are immediately killed** and your shell prompt is returned.
- Here's an example of the use of Control-C:

```
$ man chmod
```

```
CHMOD(1V)  USER  COMMANDS  CHMOD(1V)  
NAME
```

```
chmod - change the permissions mode of a file
```

```
^C
```

```
$ _
```

Pausing Output to Terminal: Control-s/Control-q

- If the output of a process starts to rapidly scroll up the screen, you may **pause** it by **pressing Control-S**.
- **To resume the output**, you may either **press Control-s again** or **press Control-q**.
- This sequence of control characters is sometimes called **XON/XOFF** protocol.
- Here's an example of its use:

```
$ man chmod
```

```
...
```

```
^s
```

---> suspend terminal output.

```
^q
```

---> resume terminal output.

```
...
```

```
$ _
```

End of Input: Control-d

- You must tell the utility when **the input from the keyboard is finished**.
- To do so, **press Control-D** on a line of its own after the last line of input. **Control-D** signifies the end of input.
- For example, the **mail utility** allows you to send mail from the keyboard to a named user:

\$ **mail tim** --> send mail to my friend tim.
Hi Tim, --> input is entered from the keyboard.
I hope you get this piece of mail. How about building a country
one of these days?

- with best wishes from Graham
^D --> tell the terminal that there's no more input.
\$ _

Setting/Changing password

- After you [first login to a UNIX system](#), it's a good idea [to change your initial password](#). The password protects all your private information.
- You might be forced to change your password by the administrator.
- Remember however, that [superuser can access everything](#).
- Passwords should generally [be at least six letters long](#) and should [not be words from a dictionary or proper nouns](#). Some system administrators will put restrictions on the life span of passwords, so you have to exchange them periodically.
- The best is to use a mixed expression of letters, numbers and other characters, like [“GWK#145W%”](#).
- If you [forget your password](#), the only thing to do is to [contact your system administrator](#) and [ask for a new password](#).

Setting/Changing Password: passwd

- `passwd` allows you to change your password.
- You are `prompted for your old password` and then twice for the new one.
- The new password may be stored in an encrypted form in the password file `“/etc/passwd”` or in a `“shadow”` file (for more security), depending on your version of UNIX.

- An example of changing a password:

\$ `passwd`

Current password : `penguin`

New password(? For help) : `GWK145W` --> invisible

New password(again) : `GWK145W` --> invisible

Password changed for glass

\$ _

- Note that you `wouldn't normally be able to see the passwords`, as UNIX turns off the keyboard `echo` when you enter them.

Logging out

- To leave the UNIX system, press the keyboard sequence **Control-D** at your shell prompt.
- This command tells your login shell that there is no more input for it to process, causing it to disconnect you from the UNIX system.
- Most systems then display a “login:” prompt and wait for another user to log in.
- \$ ^D
- UNIX® System V Release 4.0
- login : --> wait for another user to log in.
- If you connect to a remote server through ssh connection, you will not see the new login prompt; instead, you will be disconnected.

User Home Directory

- Every UNIX process has a location in the directory hierarchy, termed its current working directory.
- When you log into a UNIX system, your shell starts off in a particular directory called your home directory.
- In general, every user has a different home directory, which often begins with the prefix “/home”.
- For example, my author’s home directory is called “/home/glass”.
- The system administrator assigns these home-directory values.

Printing Working Directory: pwd

- To display **your shell's current working directory**, use the **pwd** utility, which works like this:

UNIX® System V Release 4.0

login : **glass**

Password :

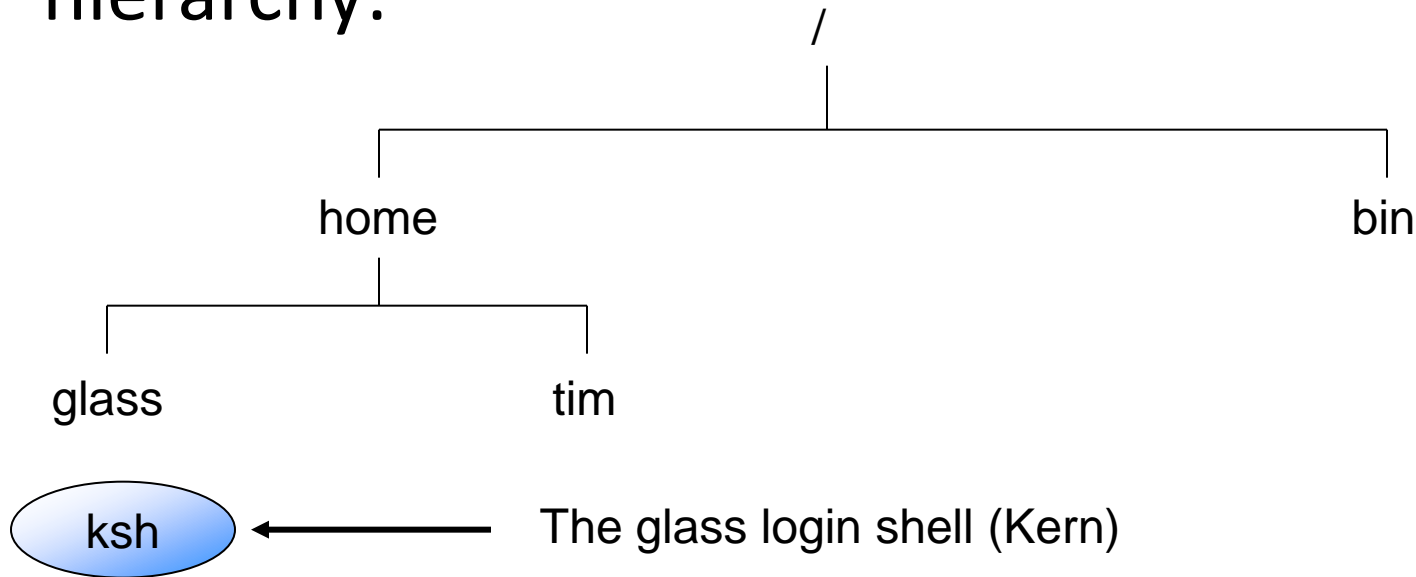
\$pwd

/home/glass

\$ _

Directory Hierarchy, Home Directory and Login Shell

- Here's a diagram that indicates the location of user's login Korn shell in the directory hierarchy:



Creating a file with cat

`cat -n fileName`

- The cat utility takes its input from standard input or from a list of files and displays them to standard output.
- The `-n` option adds line numbers to the output. cat is short for “concatenate” which means “to connect in a series of links.”
- By default, the standard input of a process is from the keyboard and the standard output is to the screen.

`$ cat > heart` --> store keyboard input into a file called “heart”.

I hear her breathing,
I'm surrounded by the sound.
Floating in this secret place,
I never shall be found.

`^D` --> tell cat that the end of input has been reached.

`$ _`

Listing Contents of a Directory: ls

- The **ls** utility, which lists information about a file or a directory.

ls -adlsFR fileName or directoryName

- **ls** lists all of the files in the current working directory in alphabetical order, excluding files whose names start with a period.
- The **-a** option causes such files to be included in the listing.
- The **-d** option causes the details of the directories themselves to be listed, rather than their contents.
- The **-g** option list a file's group.
- The **-l** option generates a long listing, including permission flags, the file's owner, and the last modification time.

Listing Contents of a Directory

- The `-s` option causes the number of disk blocks that the file occupies to be included in the listing. (A block is typically between 512 and 4K bytes.)
- The `-F` option causes a character to be placed after the file's name to indicate the type of the file:
 - `*` means an executable file, `/` means a directory file,
 - `@` means a symbolic link, and `=` means a socket.
- The `-R` option recursively lists the contents of a directory and its subdirectories.

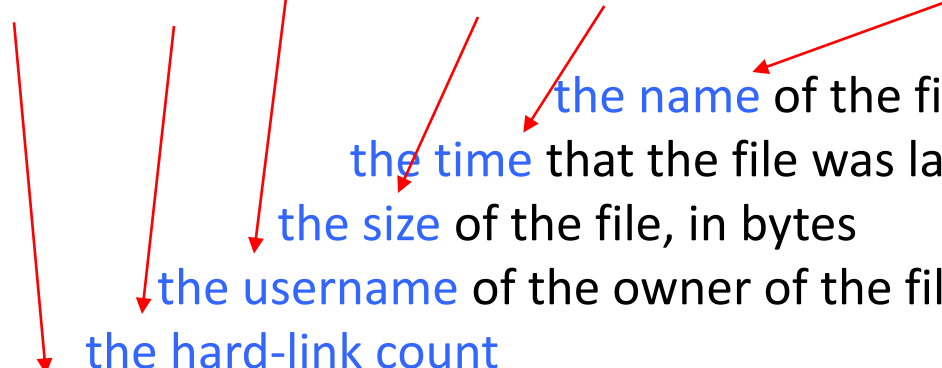
Directory Listing

- Here's an example of the use of `ls` :

`$ ls` --> list all files in current directory.
heart

`$ ls -l heart` --> long listing of "heart."
-rw-r--r-- 1 glass 106 Jan 30 19:46 heart

`$ _`



the name of the file
the time that the file was last modified
the size of the file, in bytes
the username of the owner of the file
the hard-link count
permission mode of the file

Directory Listing

Field #	Field value	Meaning
1	-rw-r--r--	the type and permission mode of the file, which indicates who can read, write, and execute the file
2	1	the hard-link count
3	glass	the username of the owner of the file
4	106	the size of the file, in bytes
5	Jan 30 19:46	the time that the file was last modified
6	heart	the name of the file

Listing Contents of a Directory

- You may obtain **even more information** by using additional options:

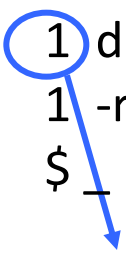
\$ **ls -algFs**

--> extra-long listing of current dir

total 3

--> total number of blocks of storage.

1	drwxr-xr-x	3	glass	cs	512	Jan	30	22:52	./
1	drwxr-xr-x	12	root	cs	1024	Jan	30	19:45	../
1	-rw-r--r--	1	glass	cs	106	Jan	30	19:46	heart

\$ _


- The **-s option** generates an extra first field, which tells you **how many disk blocks** the file occupies.
- On some UNIX systems, **each disk block is 1024 bytes long**, which implies that a 106-byte file **actually takes up 1024 bytes** of **physical storage**.

Displaying a File: cat

- `cat` with the name of the file that you wanted to display:

\$ `cat heart` --> list the contents of the “heart”
file.

I hear her breathing.

I’m surrounded by the sound.

Floating in this secret place,

I never shall be found.

\$ _

- `cat` is good for listing the contents of small files, but it doesn’t pause between full screens of output.

Displaying a File: more

`more -f +lineNumber fileName`

- The more utility allows you to scroll a list of files, one page at a time.
- By default, each file is displayed starting at line 1, although the +option may be used to specify the starting line number.
- The -f option tells more not to fold (or wrap) long lines.
- After each page is displayed, more displays the message “--more--” to indicate that it’s waiting for a command.
- To list the next page, press the space bar.
- To list the next line, press the Enter key.
- To quit from more, press the “q” key.
- ^B will display the previous page
- H will display help page
- Try:

```
$ ls -la /usr/bin > myLongFile
```

```
$ more myLongFile
```

Displaying a File: head and tail

head -n fileName

- The head utility **displays the first n lines of a file**. If n is not specified, it defaults to 10. If more than one file is specified, **a small header identifying each file** is displayed **before its contents**.

tail -n fileName

- The tail utility **displays the last n lines of a file**. If n is not specified, it defaults to 10. If more than one file is specified, **a small header identifying each file** is displayed before its contents.
- The first two lines and last two lines of my “heart” file.

\$ head -2 heart

--> list the first two lines.

I hear her breathing,
I'm surrounded by the sound.

\$ tail -2 heart

--> list the last two lines.

Floating in this secret place,
I never shall be found.

\$ head -15 myLongFile

--> see what happens

Renaming/Moving a File: mv

`mv -i oldFileName newFileName`

`mv -i fileName directoryName`

`mv -i oldDirectoryName newDirectoryName`

- The first form of mv **renames oldFileName as newFileName**.
- The second form allows you **to move a collection of files to a directory**.
- The third form allows you **to move an entire directory**.
- **The -i option** prompts you **for confirmation** if newFileName already exists so that you **do not accidentally replace its contents**. You should learn to use this option (or set a convenient shell alias that replaces “mv” with “mv -i”; we will come back to this later).

Renaming/Moving Files: mv

- Here's how to **rename the file** using the first form of the mv utility:

```
$ mv heart heart.ver1 -->  
    rename to "heart.ver1".
```

```
$ ls
```

```
heart.ver1
```

```
$ _
```

Making Directory: mkdir

`mkdir -p newDirectoryName`

- The mkdir utility **creates a directory**. The **-p option** creates any parent directories in the newDirectoryName pathname **that do not already exist**.
- If newDirectoryName **already exists**, an **error message is displayed** and the existing file is not altered in any way.

```
$ mkdir lyrics           --> creates a directory called "lyrics".  
$ ls -lF                --> check the directory listing in order  
                        --> to confirm the existence of the  
                        --> new directory.
```

```
-rw-r--r--    1  glass  106  Jan 30 23:28  heart.ver1  
drwxr-xr-x    2  glass  512  Jan 30 19:49  lyrics/  
$ _
```

Moving Files

- Once the “lyrics” directory is created, we can move the “heart.ver1” file into its new location. To do so, used `mv` and confirm the operation using `ls`:

```
$ mv heart.ver1 lyrics
```

--> move into “lyrics”.

```
$ ls  
lyrics/
```

--> list the current directory.

--> “heart.ver1” has gone.

```
$ ls lyrics  
heart.ver1  
$_
```

--> list the “lyrics” directory.

--> “heart.ver1” has moved.

Changing Directories: cd

- `cd directoryName`

- The following might be inconvenient; especially if we deal with large hierarchy:

\$ `vi lyrics/heart.ver1` --> invoke the vi editor

- Instead, change directory:

\$ `cd lyrics` --> change directory

\$ `vi heart.ver1` --> invoke the vi editor

- The `cd shell command` changes a shell's `current working directory` to be `directoryName`.
- If the `directoryName` argument is omitted, the shell is moved to its `owner's home directory`.

Reorganizing Directories

```
$ pwd
```

--> display where I am

```
/home/glass
```

```
$ cd lyrics
```

--> move into the “lyrics” directory

```
$ pwd
```

```
/home/glass/lyrics
```

```
$ cd ..
```

--> move up one level

```
$ pwd
```

--> display new position

```
/home/glass
```

```
$ cd lyrics
```

--> move into the “lyrics” directory

```
$ pwd
```

```
/home/glass/lyrics
```

```
$ ls ~/
```

--> “~/” refers to home directory

```
/home/glass
```

```
$ _
```

Copying Files: cp

- To copy the file, I used **the cp utility**, which works as follows:

cp -i oldFileName newFileName

cp -ir fileName directoryName

- The first form of cp **copies the contents of oldFileName** to newFileName.
- If the label newFileName **already exists**, its contents are **replaced by** the contents of oldFileName.
- **The -i option** prompts you **for confirmation** if newFileName already exists so that you do not accidentally overwrite its contents. Like with **mv**, it is a good idea to use this option or create an alias.

Copying Files: cp

- The `-r` option causes any source files that are directories to be recursively copied, thus copying the entire directory structure.
 - `cp` actually does two things
 - It makes a physical copy of the original file's contents.
 - It creates a new label in the directory hierarchy that points to the copied file.
- `$ cp heart.ver1 heart.ver2` --> copy to "heart.ver2".
- `$ ls -l heart.ver1 heart.ver2` --> confirm the existence of both files.
- | | | | | | | | |
|------------|---|-------|-----|-----|----|-------|------------|
| -rw-r--r-- | 1 | glass | 106 | Jan | 30 | 23:28 | heart.ver1 |
| -rw-r--r-- | 1 | glass | 106 | Jan | 31 | 00:12 | heart.ver2 |
- `$ cp -i heart.ver1 heart.ver2` --> what happens?

Deleting a Directory: rmdir

rmdir directoryName

- The rmdir utility **removes all of the directories** in the list of directory names provided in the command. A directory must **be empty before it can be removed**.
- **To recursively remove a directory** and all of its contents, use **the rm utility** with the **-r option**.
- Here, we try **to remove the “lyrics.draft” directory** while it still contains the draft versions, **so we receive the following error message**:

```
$ rmdir lyrics.draft
```

```
rmdir : lyrics.draft : Directory not empty.
```

```
$ _
```


Deleting Directories: rm -r

- The rm utility allows you to remove a file's label from the hierarchy.
- Here's a description of rm:

rm -fir fileName

- The rm utility removes a file's label from the directory hierarchy.
- If the filename doesn't exist, an error message is displayed.
- The -i option prompts the user for confirmation before deleting a filename. It is a very good idea to use this option or create a shell alias that translates from "rm" to "rm -i". If you don't, you will lose some files one day – you have been warned!
- If fileName is a directory, the -r option causes all of its contents, including subdirectories, to be recursively deleted.
- The -f option inhibits all error messages and prompts. It overrides the -i option (also one coming from an alias). This is dangerous!

Removing Directories with Files

- The `-r` option of `rm` can be used to delete the “lyrics.draft” directory and all of its contents with just one command:

\$ `cd` --> move to my
home directory.

\$ `rm -r lyrics.draft` --> recursively
delete directory.

\$ _

Printing Files: lp, lpstat and cancel

lp [-d destination] [-n copies] {fileName}

- **lp** prints the named file(s) to the printer specified by **the -d option**. You get the name of the printer from the system administrator.
- If no files are specified, standard input is printed instead.
- By default, **one copy of each file is printed**, although this default may be overridden by using **the -n option** to specify the number of copies.

lpstat [destination]

- **lpstat** displays the status of all print jobs sent to any printer with **the lp command**.
- If a printer destination is specified, **lpstat reports queue information for that printer only**.
- **lpstat** displays information about the user, the name and size of the job, and a print-request ID.

Canceling a Print Job: cancel

cancel **request-ID**

- **cancel** removes all of the specified jobs from the printer queue.
- If you're a super-user, then you may cancel any queued job, even if it was ordered by someone else.

\$ **lp -d lwcs heart.final**

--> order a printout.

request id is lwcs-37(1 file)

\$ **lpstat lwcs**

--> look at the printer status.

printer queue for lwcs

lwcs-36 ables priority 0 Mar 18 17:02 on lwcs inventory.txt 457 bytes

lwcs-37 glass priority 0 Mar 18 17:04 on lwcs heart.final

\$ **cancel lwcs-37**

--> look at the printer status.

\$ _

Print Files (BSD, Linux): lpr, lpq and lprm

lpr -m [-Pprinter] [-#copies] {fileName}

- **lpr** prints the named file(s) to the printer specified by **the -P option**. If no printer is specified, then the value of **\$PRINTER** environment variable is used.
- **If no files are specified**, standard input is printed instead.
- By default, **one copy of each file is printed**, although this default may be overridden by using **the -# option** to specify the number of copies.
- To receive mail when the printing job is done, **-m** option is used.

Querying Print Jobs: lpq

`lpq -l [-Pprinter] {job#} {userId}`

- `lpq` displays the status of all print jobs sent to the printer referenced by the `-P` option (or `$PRINTER` environment variable there is no `-P` option). The scope can be limited to just specified jobs and/or users.
- `lpq` displays information about the user, the name and size of the job, and a print-request ID.
- The `-l` option can be used to generate extra information.

Removing Print Jobs: lprm

lprm [-Pprinter] [-] {jobs} {userId}

- **lprm** removes all of the specified jobs from the queue of the printer specified by the **-P** option (or by **\$PRINTER** environment variable if no printer is specified). The scope of the command can be controlled by specifying jobs to remove.
- The **-** option removes all jobs requested by the user issuing the command.
- If you're a super-user, then you may cancel any queued job, even if it was ordered by someone else by specifying the user Id.

\$ lpr -Plwcs heart.final

--> order a printout

request id is lwcs-37(1 file)

\$ lpq -Plwcs glass

--> look at the printer status

lwcs is ready and printing

Rank	Owner	Job	File(s)	Total Size
1st	glass	25	heart.final	106 bytes

\$ lprm -Plwcs 25 glass

--> remove the job

Counting Lines, Words and Characters in Files: wc

wc -lwc fileName

- The wc utility counts the number of lines, words, and/or characters in a list of files.
- If no files are specified, standard input is used instead.
- The -l option requests a line count,
- the -w option requests a word count,
- and the -c option requests a character count.
- If no options are specified, then all three counts are displayed.
-
- A word is defined by a sequence of characters surrounded by tabs, spaces, or new lines.

Counting Lines, Words and Characters in Files: wc

- For example, to count lines, words and characters in the “heart.final” file, we used:

```
$ cd ~/lyrics.final
```

```
$ wc heart.final
```

--> obtain a count of
the number of lines,
--> words, and
characters.

```
9 43 213 heart.final
```

```
$ _
```

Determining Type of a File: file

file fileName

- The `file` utility attempts to describe the contents of the `fileName` argument(s), including the language in which any of the text is written.
- `file` is not reliable; it may get confused.
- When `file` is used on a symbolic-link file, `file` reports on the file that the link is pointing to, rather than on, the link itself.
- For example,

```
$ file heart.final
heart.final: ascii text
$_
```

--> determine the file type.