Mark Stenglein, MIP 280A4

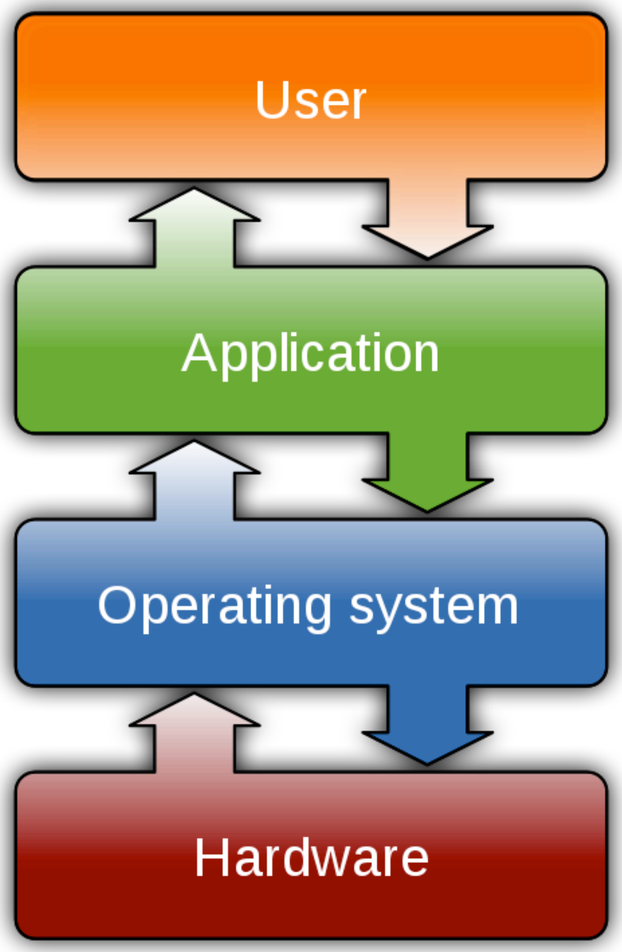
Advantages of command line computing

- Lots of essential software doesn't have graphical user interfaces (GUIs)
- The command line provides you an opportunity to take advantage of more powerful computers (servers, the cloud, …)
- Once you are comfortable in a command line environment you can work efficiently and make fast progress
- Good way to create automated and reproducible workflows
- Complements GUI-based software like Geneious, Excel, and R. Often computational workflows involve going back and forth between the command line and graphical programs
- Useful for computing in general: not at all specific to biology or science

Disadvantages of command line computing

- It is confusing at first / learning curve
- GUI-based software is easy to use and intuitive
- Visualization of results / data is valuable

A computer's operating system (OS) is the software that provides an interface between the hardware and the applications and user



Common Operating systems:

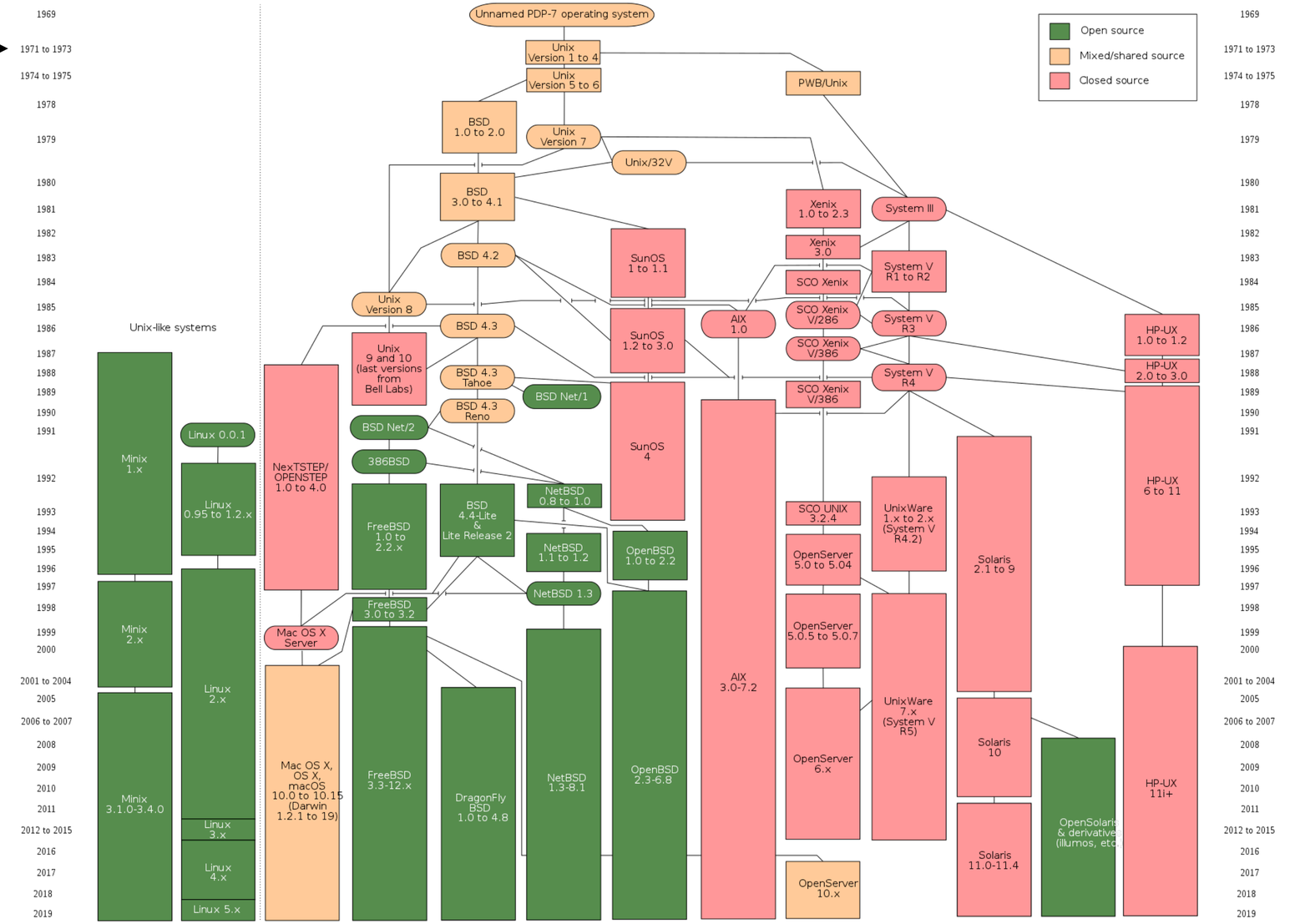
- Windows
- Mac OS
- Linux
- Chrome OS
- iOS
- Android

A lot of scientific command-line computing is done in Linux

Image: Golftheman, CC-BY-SA-3.0. Link

Linux is one of many UNIX-like operating systems

1970 →



Android ← Linux Mac-OS

Image: Eraserhead1, Infinity0, Sav_vas CC-BY-SA-3.0, link

Anatomy of a command-line command

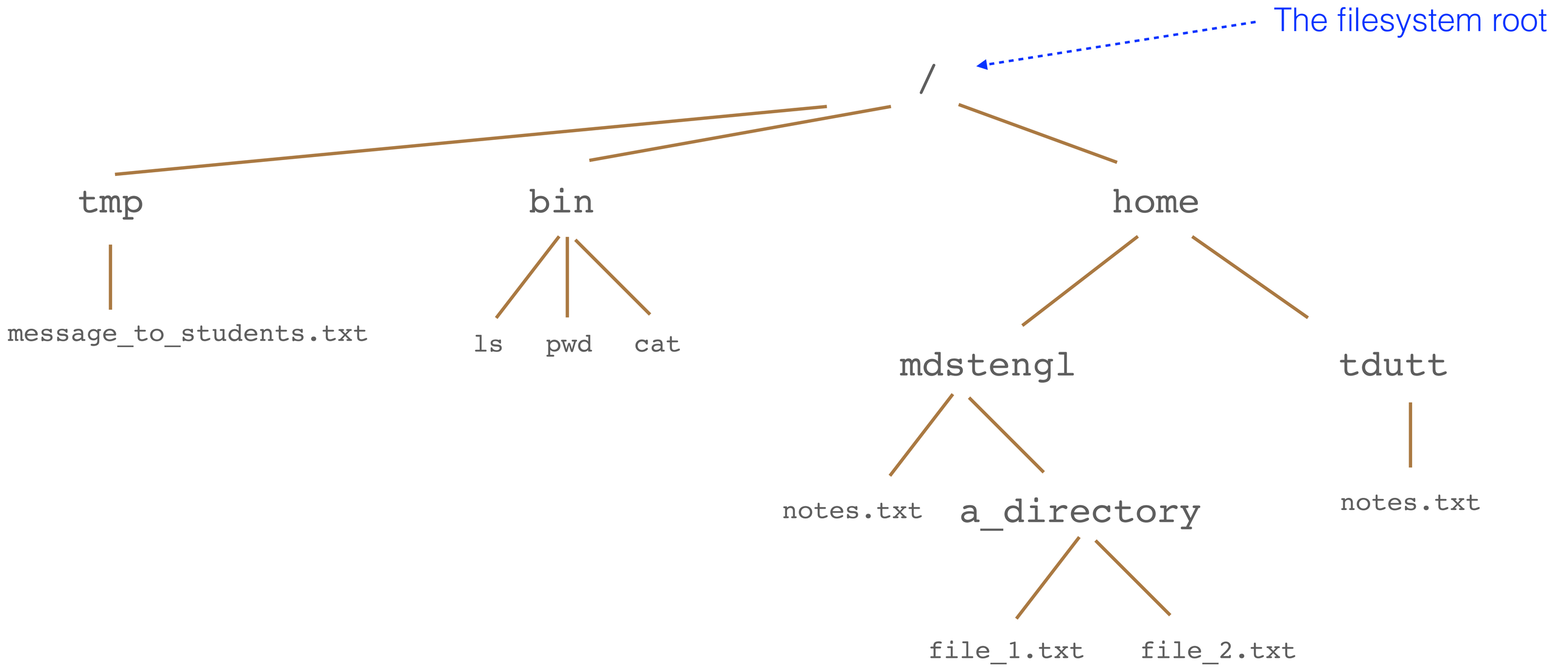
`command_name` *`one_or_more_optional_arguments`*

always

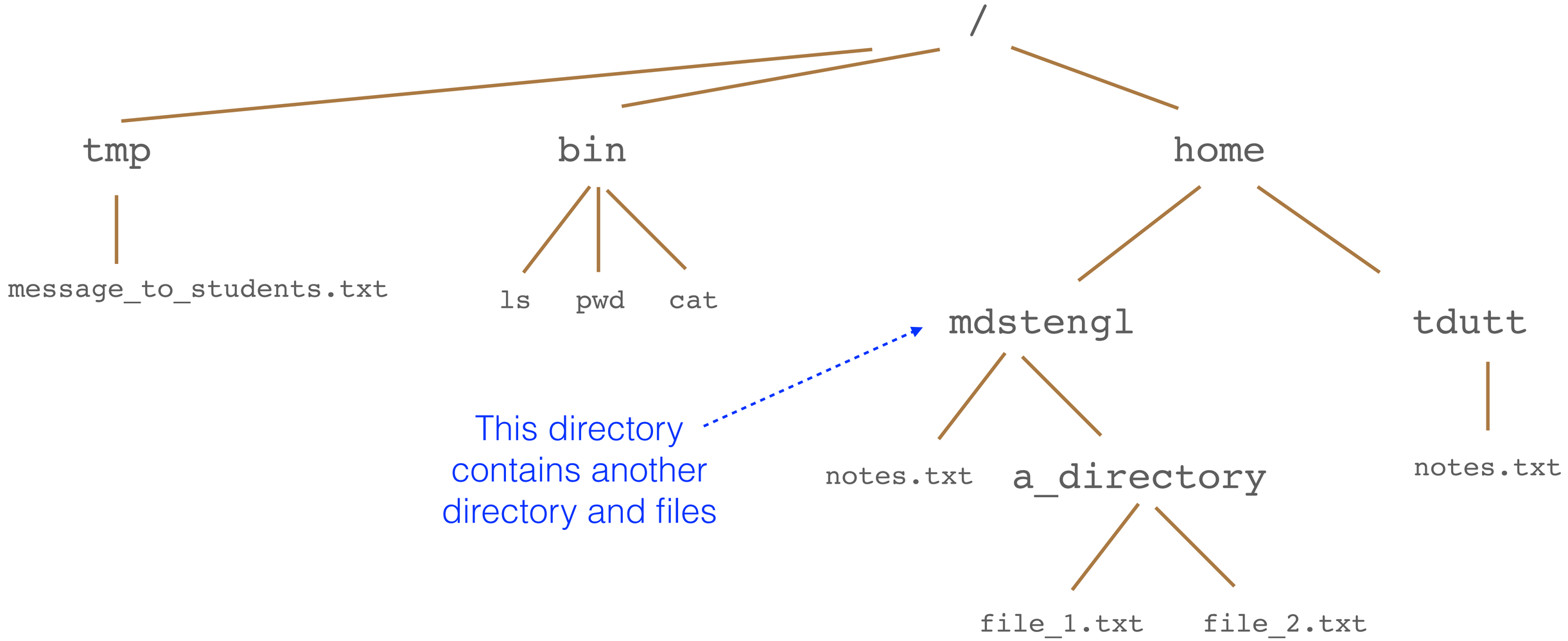
sometimes

What are some examples of commands and their options?

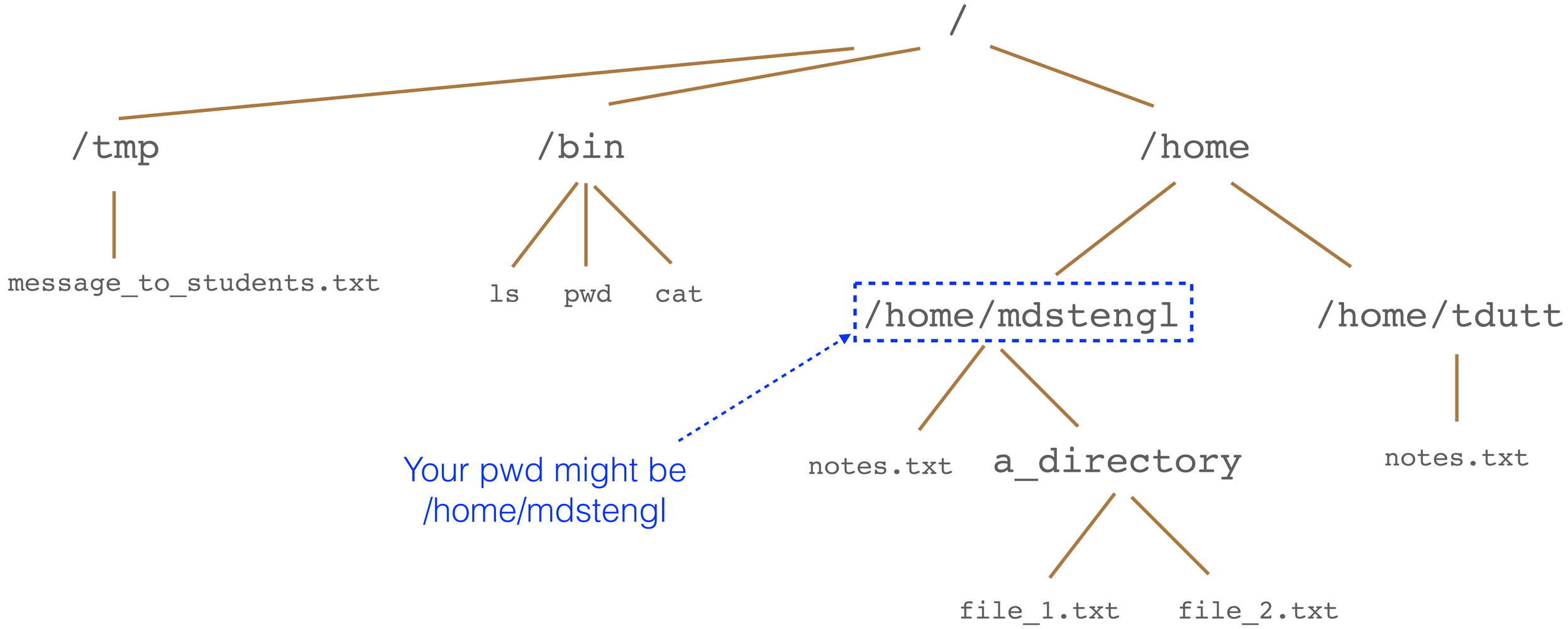
Linux filesystems are trees of directories (folders) and files



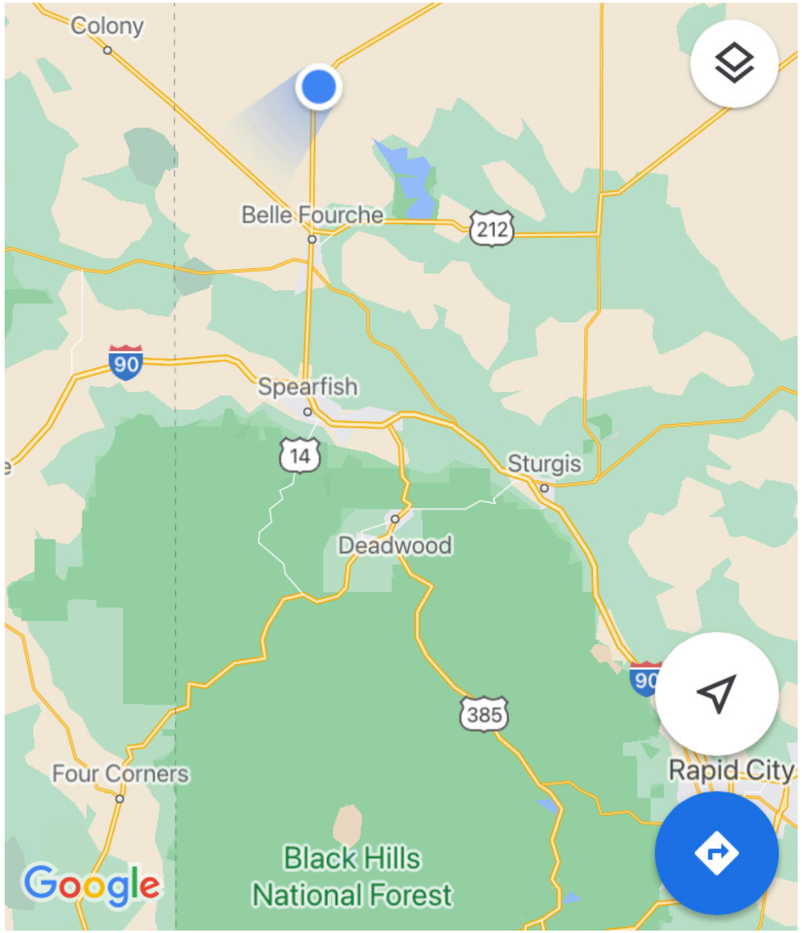
Directories contain files and other directories



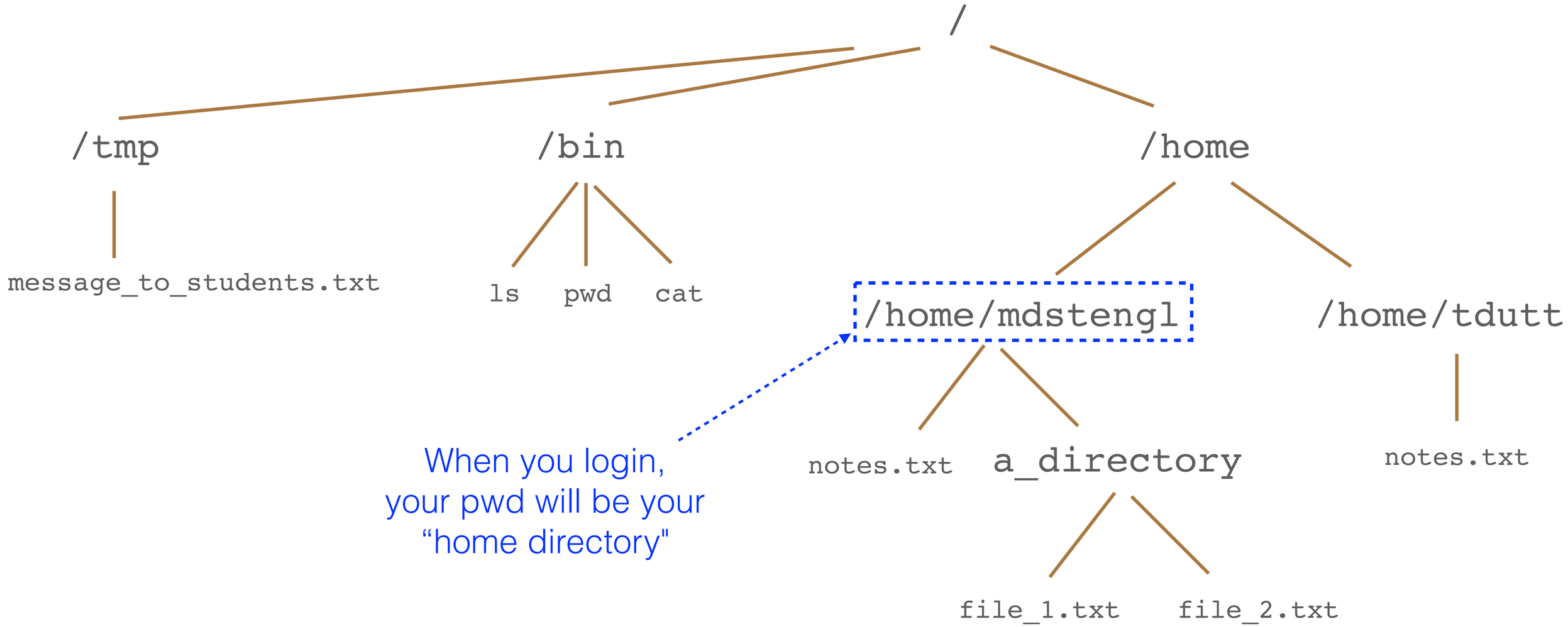
In a command line environment, you are always located in a directory known as your *present working directory* (pwd)



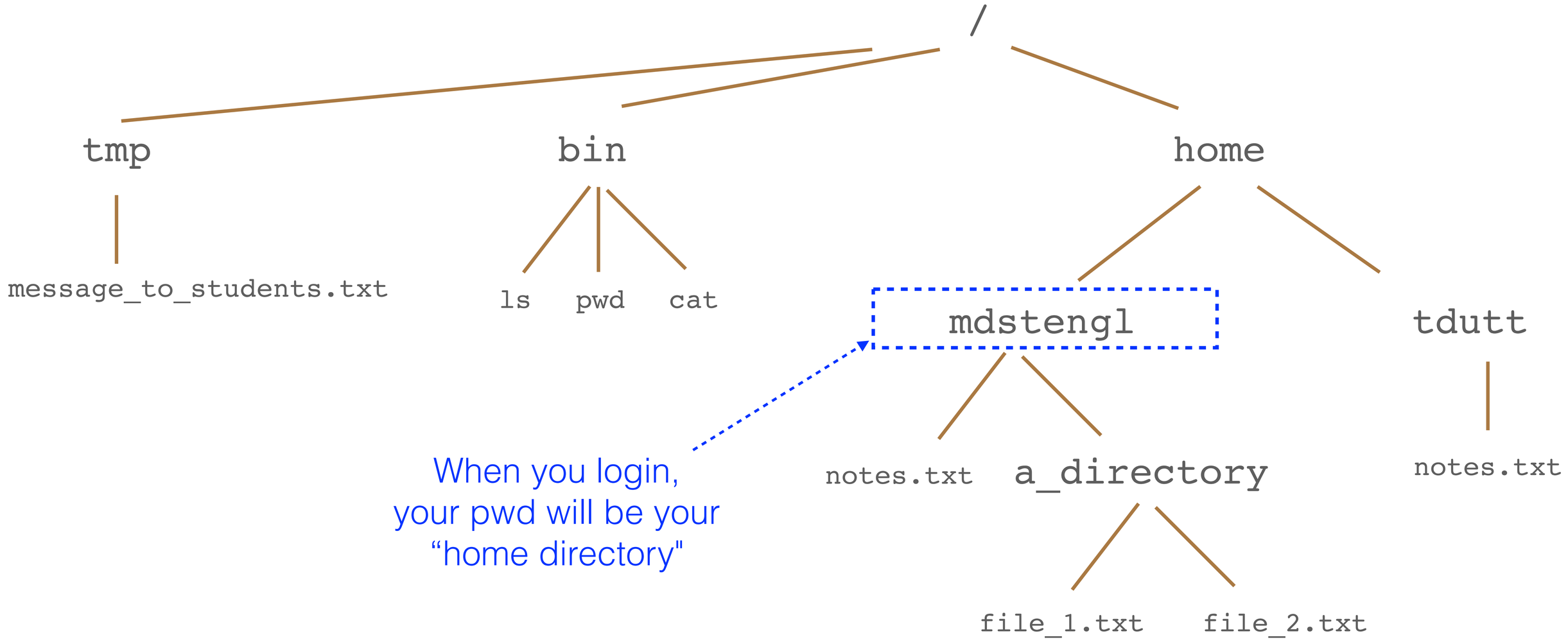
It’s important to remember that you are always somewhere when working on the command line



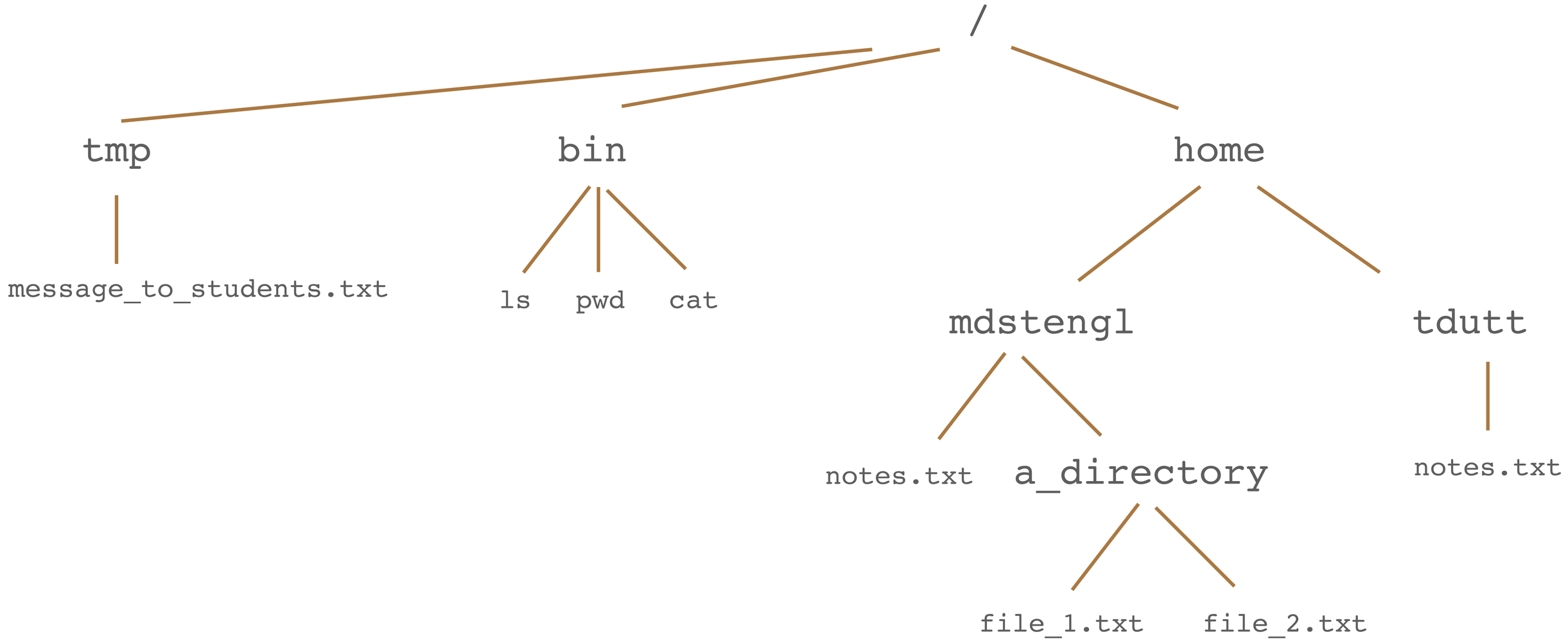
You start out at home



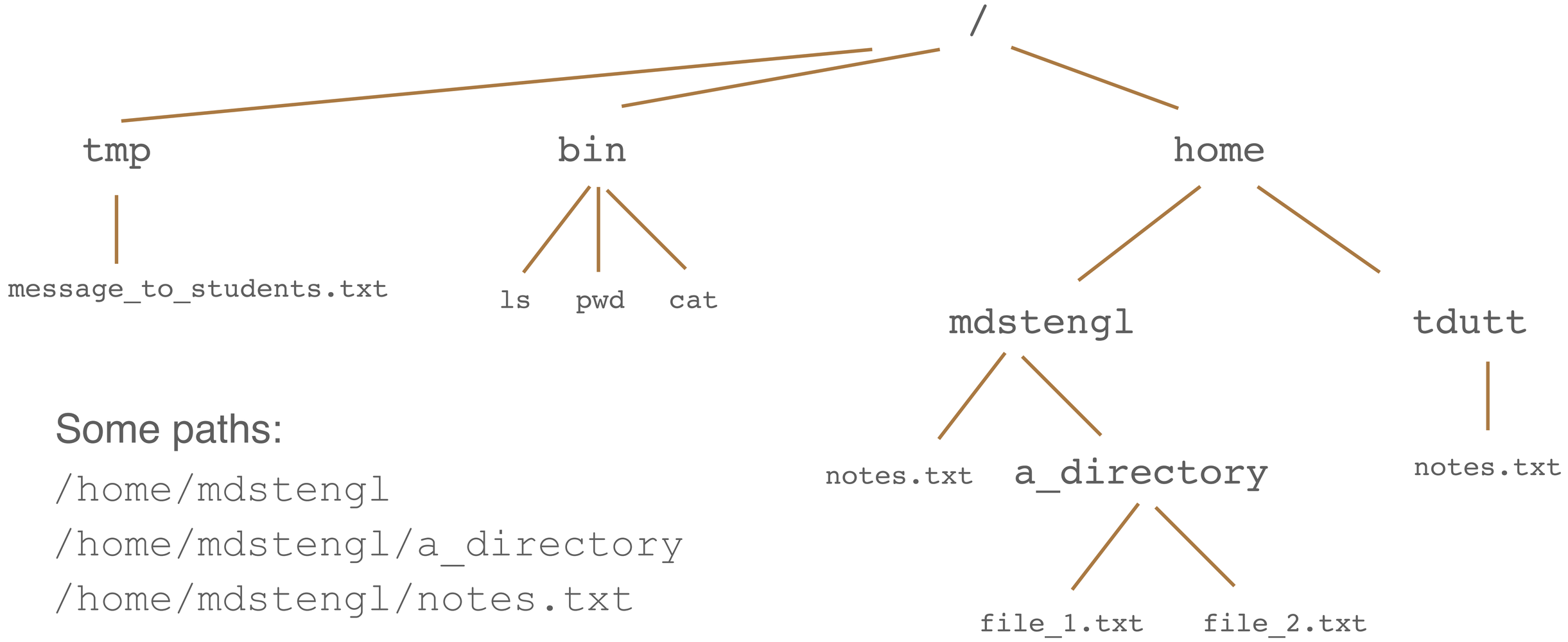
Exercise: connect (ssh) to thoth01 and determine your initial present working directory by running the `pwd` command



A path is the location of a file or a directory



A path is the location of a file or a directory



Some paths:

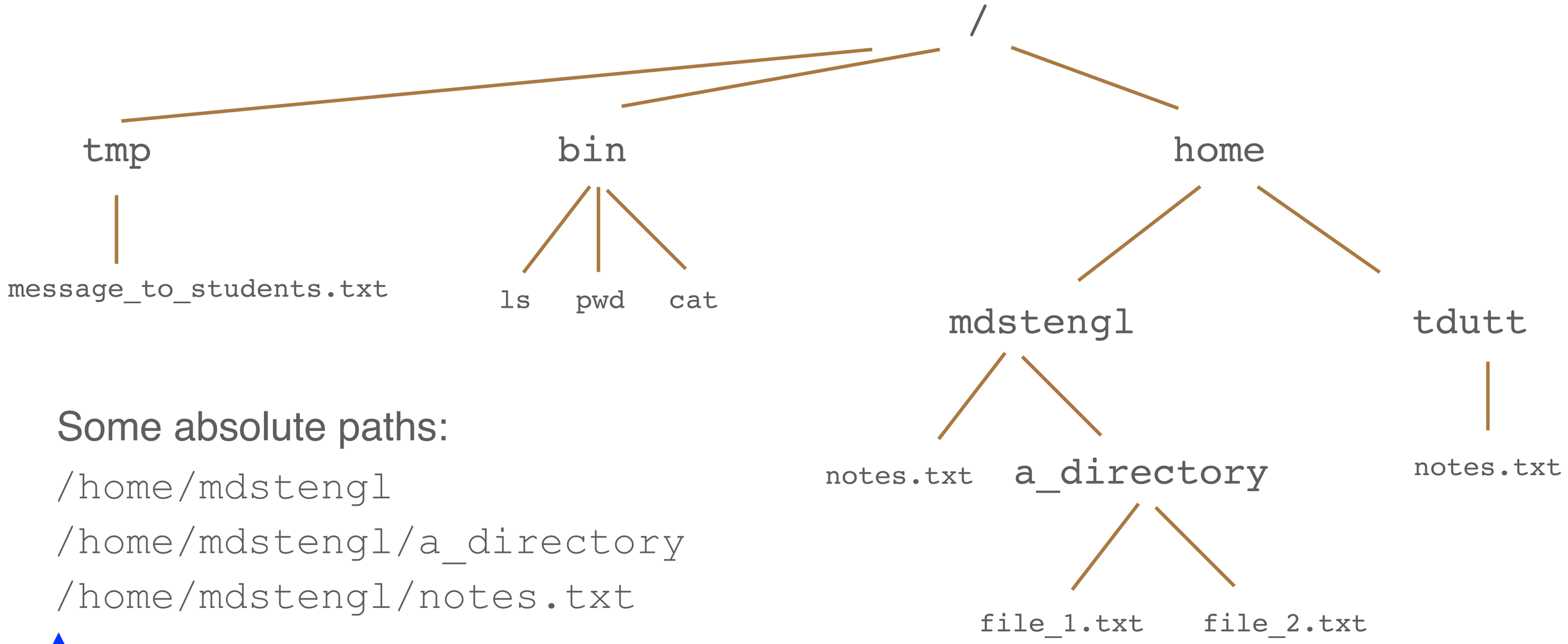
/home/mdstengl

/home/mdstengl/a_directory

/home/mdstengl/notes.txt

Paths that start at the root of the filesystem (that start with /) are absolute paths

Their meaning does not change depending on your pwd

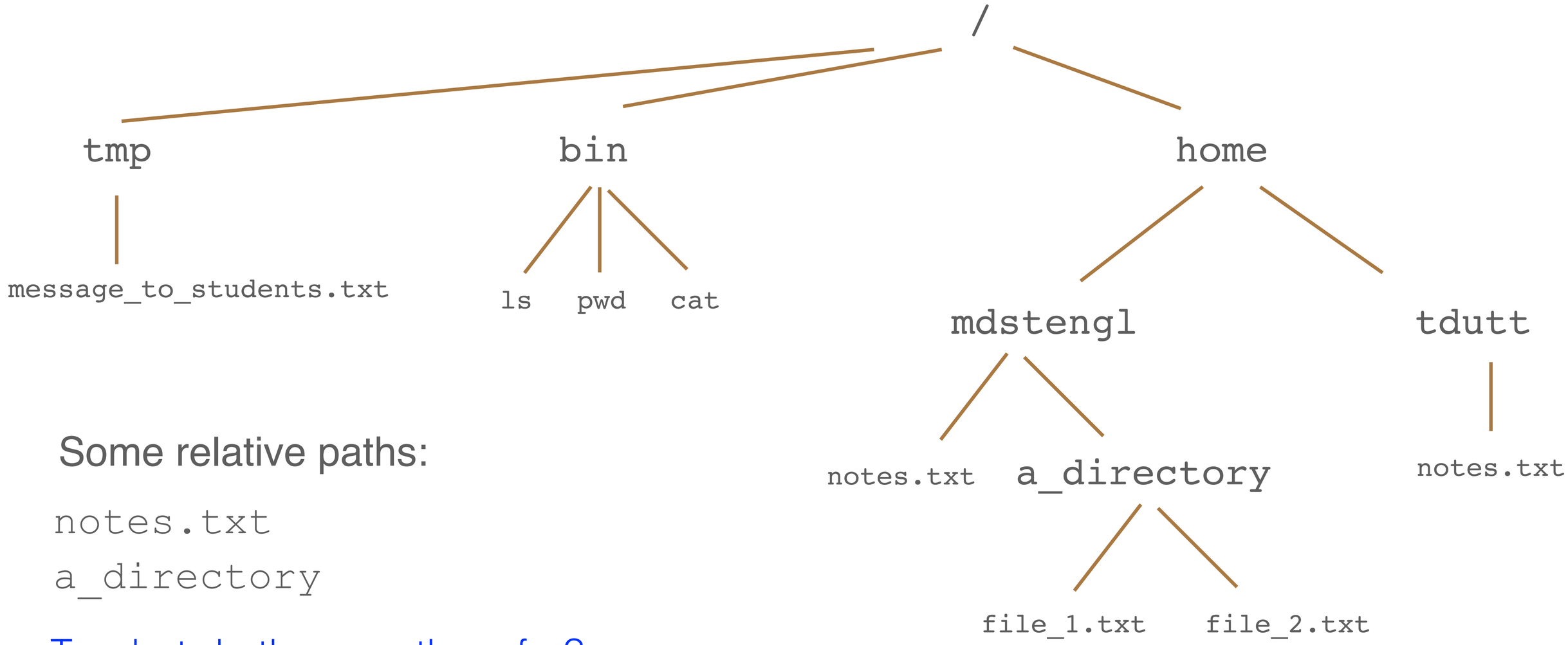


Some absolute paths:

```
/home/mdstengl
/home/mdstengl/a_directory
/home/mdstengl/notes.txt
```

absolute paths start with /

Relative paths *change meaning depending on your pwd*

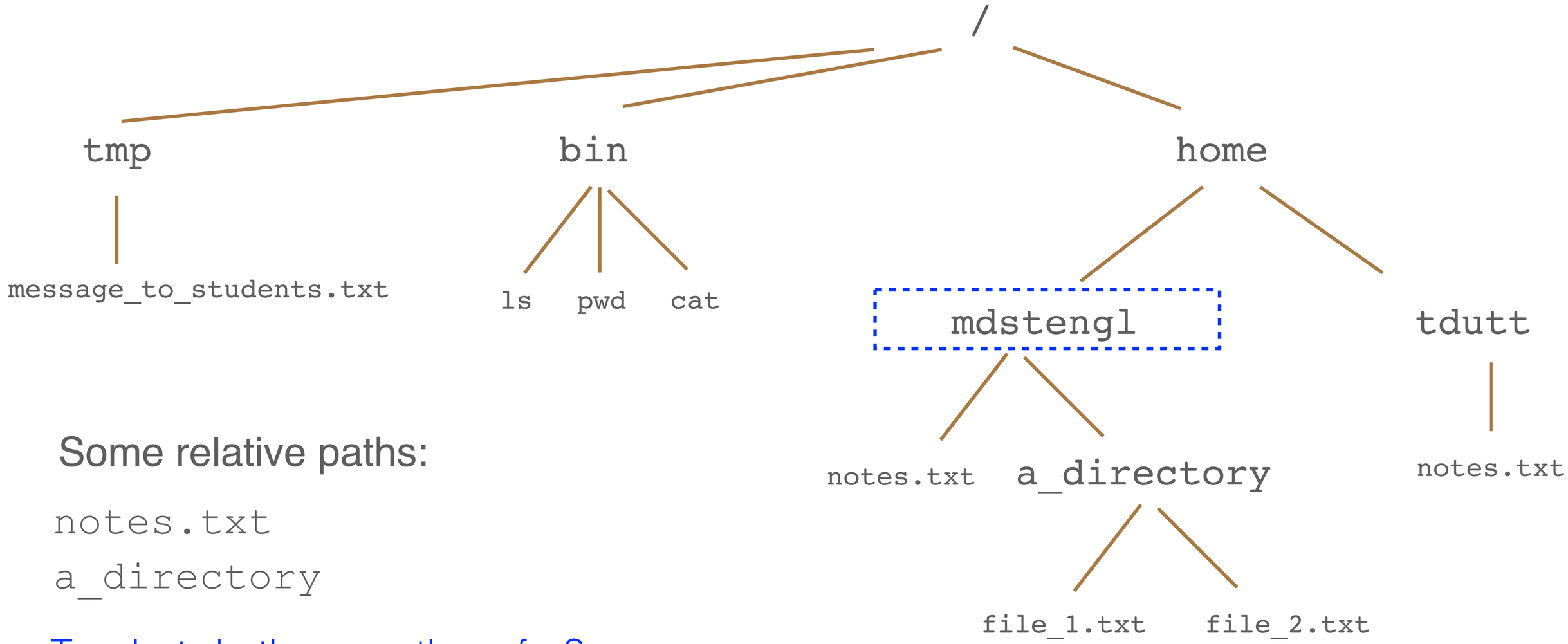


Some relative paths:

```
notes.txt
a_directory
```

To what do these paths refer?

Relative paths change meaning depending on your pwd



Some relative paths:

```
notes.txt
a_directory
```

To what do these paths refer?

Absolute and relative paths

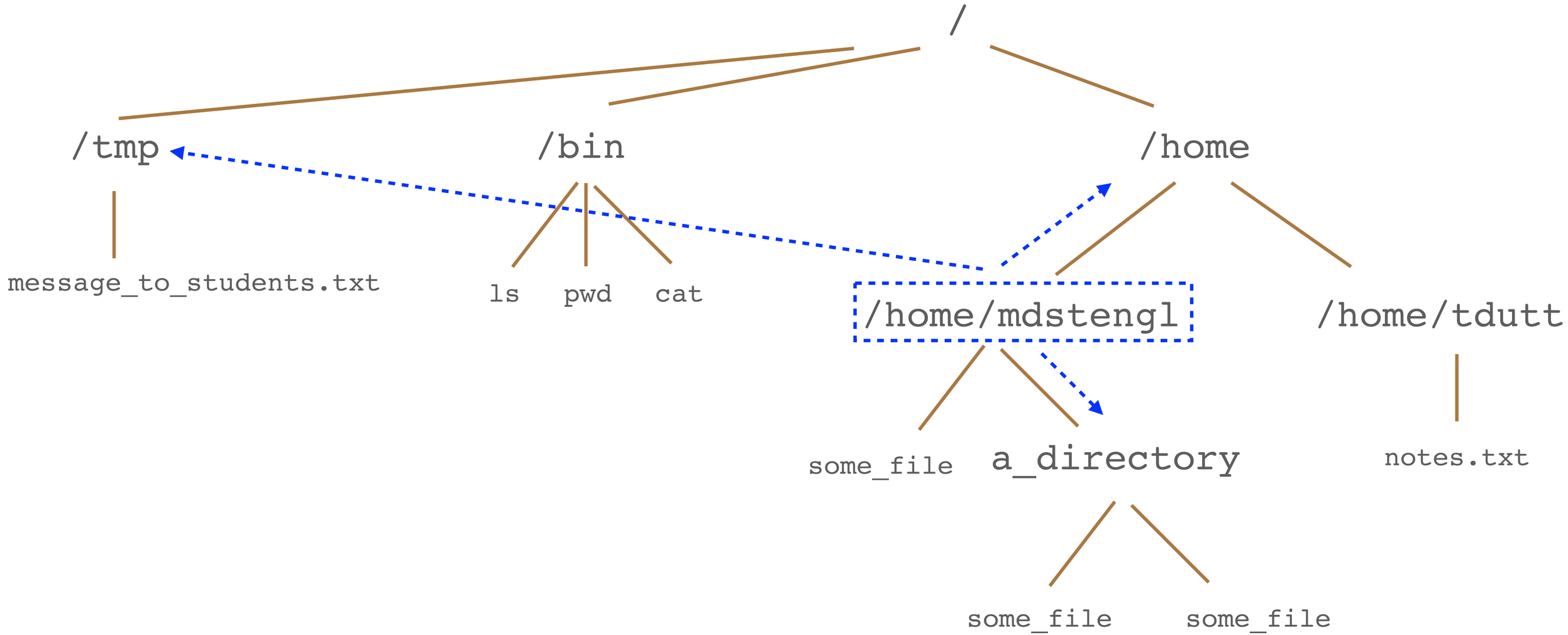
Absolute path: the meaning is independent of your pwd.
Always begins with a / (either actually or implicitly via shortcut)

Relative path: the meaning is dependent on (and relative to) your pwd.

The pwd is important because it determines the meaning of relative paths

It also defines your working space: where new files and directories will be created by default.

You can move to other directories using the change directory (cd) command



The cd command changes your pwd

```
cd path_to_change_to
```

```
cd /home
cd /home/mdstengl
cd /tmp
cd tmp
```

Note that the path can be absolute or relative

Path-related shortcuts

Shortcut	Meaning
.	Your pwd (the directory you are in)
. .	Up one directory
~	Your home directory

Path shortcuts

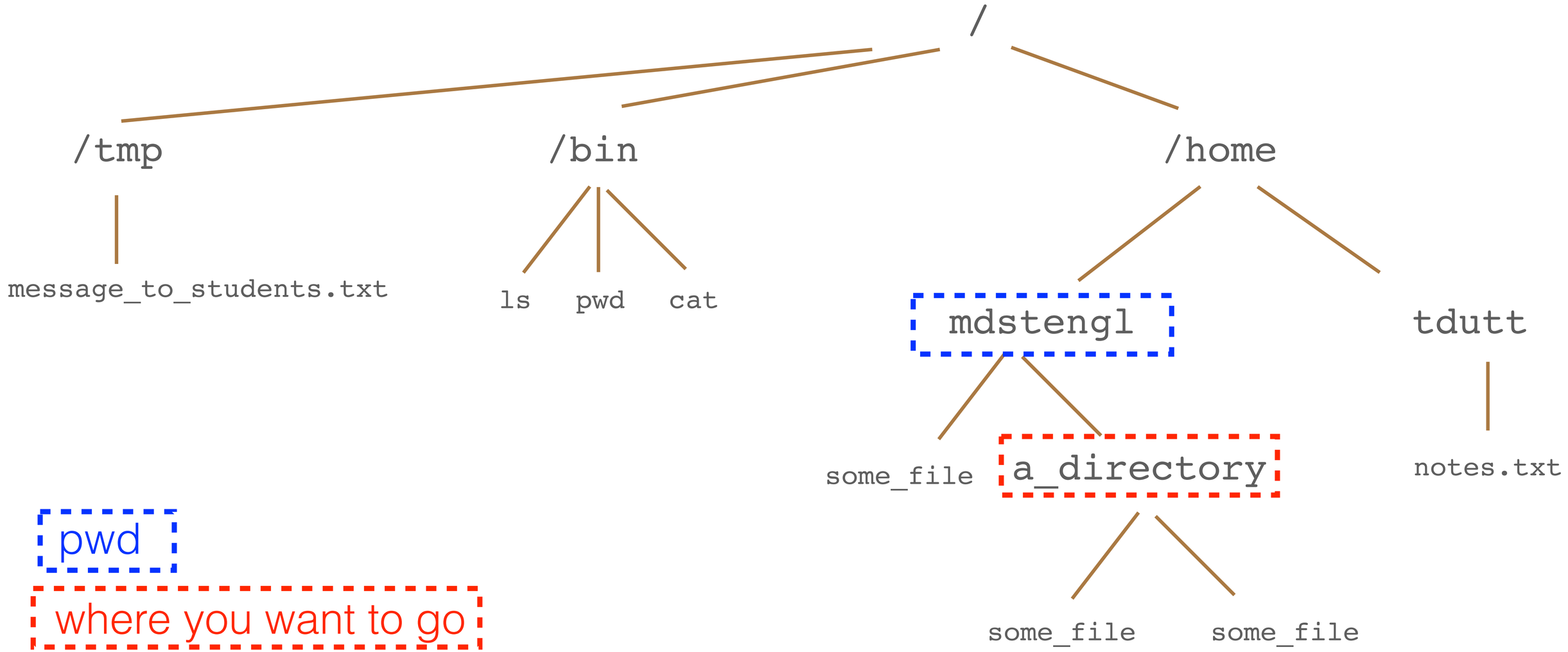
Where will these cd commands take you?

```
cd .
cd ..
cd ~
cd
```

Path shortcuts

```
cd .      Don’t actually change directory
cd ..     Go up one directory
cd ~      Go home
cd        Go home
```

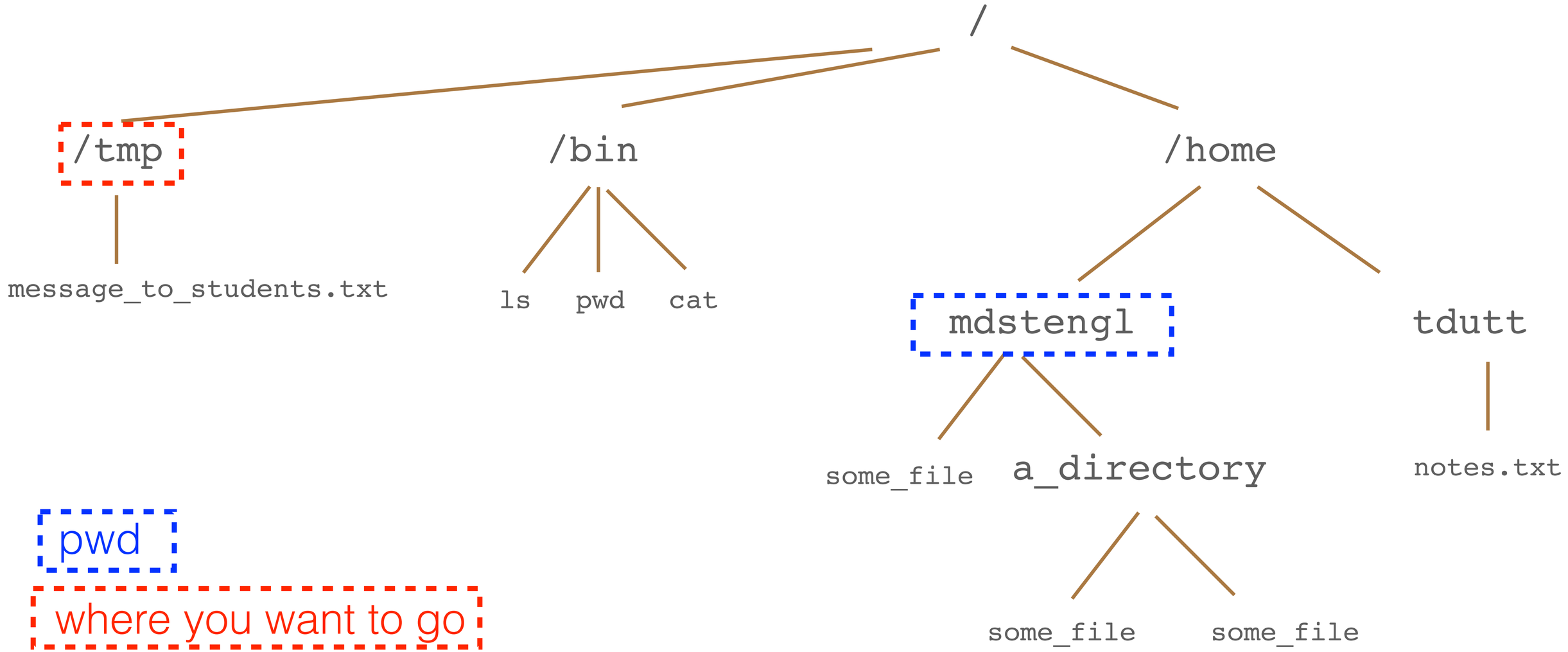
Let’s practice using cd to move around



pwd

where you want to go

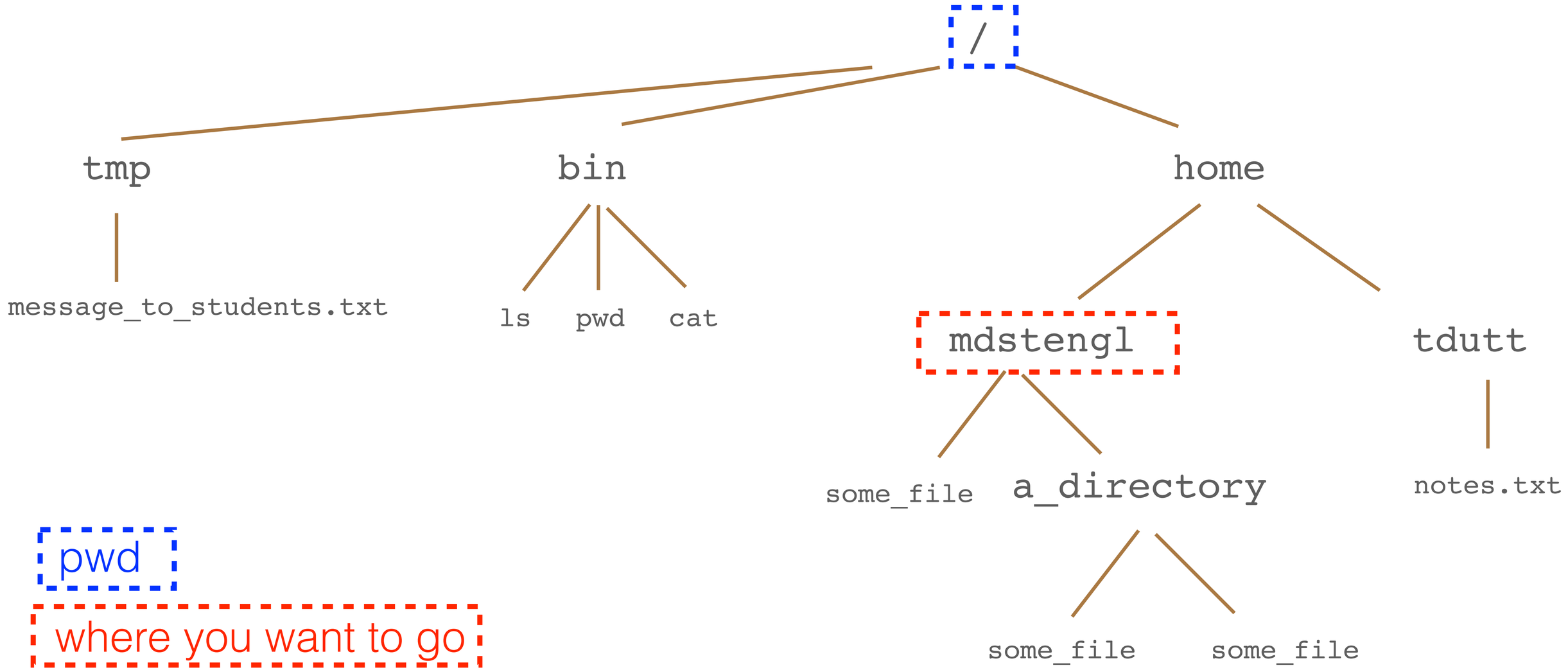
Let’s practice using cd to move around



pwd

where you want to go

Let’s practice using cd to move around



pwd

where you want to go

Exercise

- Are the following paths absolute paths or relative paths?
  - `/home/mdstengl/a_directory`
  - `~/a_directory`
  - `a_directory`
  - `../a_directory`
  - `/home/mdstengl/a_directory/..`

The `ls` command will list the files and directories in a directory

```
ls          List files and dirs in pwd
ls .        Same: remember . is the pwd
ls /bin     List contents of the /bin directory
```

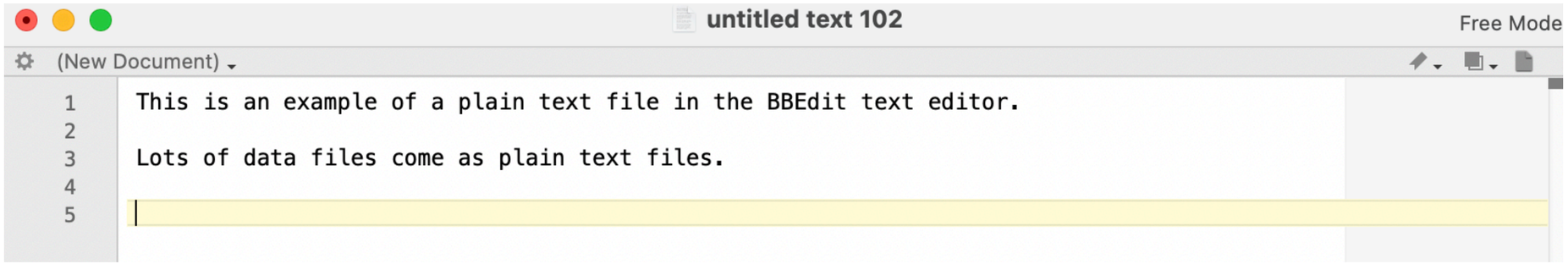
You can fine-tune most linux command by using command line options

Exercise: run the following commands on thoth01
What is different about the output of each command?

```
ls /usr/bin
ls -l /usr/bin                  # note: -l is the letter l not #1
ls -l -h /usr/bin
ls -l -h -S /usr/bin            # note: uppercase -S
ls -l -h -S -r /usr/bin
ls -lhSr /usr/bin               # often OK to mush options together
```

Files on linux are typically plain text files or binary files

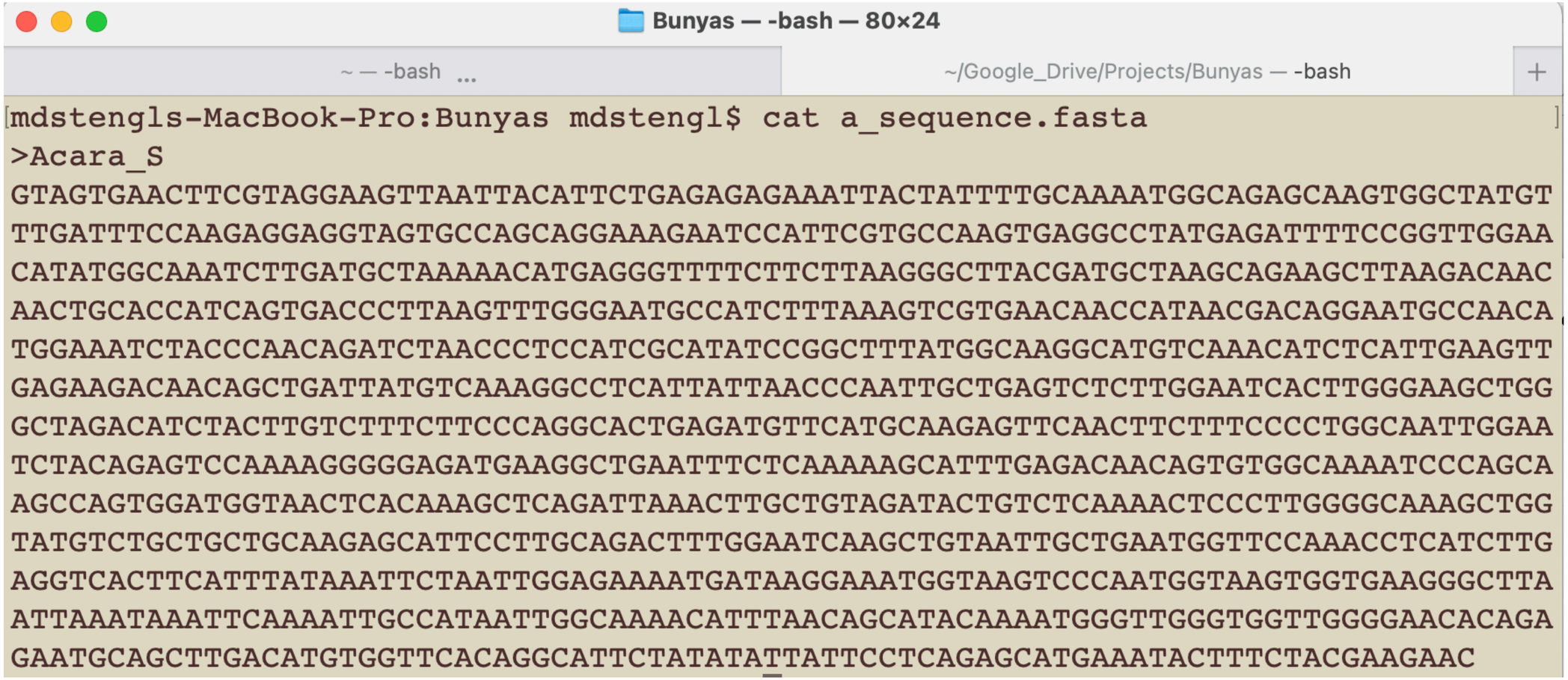
Plain text files have characters (letter) encoded by one byte
There is no formatting like in a Word document (no bold, no italics, etc.)



BBEdit is a good Mac OS plain text editor
Notepad++ is a good Windows plain text editor

Sequence data almost always comes in plain text format

A DNA sequence in “fasta” plain text format



The `cat` command will output the contents of a file

Some files, for instance programs or compressed files,
are in unreadable binary format

Exercises:

1. Use the `cat` command to output the contents of `/usr/bin/ls`

The `cat`, `less`, `head`, and `tail` commands allow you to inspect the contents of files

`cat`: outputs the contents of a file or files

`less`: pages through a file
up and down arrows, space bar, q to quit

`head`: outputs the first lines of a file
`head -N`: output the first N lines (head -5)

`tail`: outputs the last lines of a file
`tail -n N`: outputs the last N lines

Exercises: use the `cat`, `less`, `head`, and `tail` commands to inspect the contents of some files on thoth01

- Use a command that uses an absolute path to determine the secret message in `/tmp/message_1.txt`
- Use a command that uses a relative path to determine the secret message in `/tmp/message_2.txt`
- What text is on the 39th line of `/etc/update-motd.d/00-header`?

Tab completion makes your life easier!

When you press the tab key while typing a command name
The shell will try to autocomplete the command name.

When you press the tab key while typing a file or directory name, the shell will try to auto-complete the file or directory name

If there is ambiguity the shell will show you your matching options.

This helps avoid typos in command and file names!

```
Examples:
mkd<tab>
ls /home/mdste<tab>
ls /home/<tab>
```

The `touch` command will create an empty file

```
touch: create an empty file or update the last modified time
of an existing file

touch a_file.txt      # create a_file.txt or update it's
                      # last modified time if it exists
```

The `echo` command will output some text to the terminal

```
echo “this is my important message”
```

The shell has variables

These can be existing built-in variables or new ones you create.

One built-in shell variable is named `HOME`. This value of this variable is the path of your home directory.

To get the value of a shell variable, put a $ before it. Try these commands:

```
echo "My home directory is: HOME"
echo "My home directory is: $HOME"

cd /tmp
cd $HOME
```

Usually the output of shell commands goes to the terminal
But you can capture the output of a command
using the “output redirection operator”: >

```
# this output just goes to the terminal
echo “my home directory is $HOME”

# capture the output in a new file named my_home.txt
echo “my home directory is $HOME” > home.txt

# now, use cat to output the contents of this file
cat home.txt
```

The >> operator appends output to a file if it already exists
(Like > it creates a new file if it doesn't already exist)

```
# capture the output in a new file named about_me.txt
echo "my home directory is $HOME" >> about_me.txt

# append the output now to the existing file
echo "and my user name is $USER" >> about_me.txt
```

Note: A number of Linux commands behave differently
depending on whether a file already exists or not

The pipe operator `|` makes the *output* of one command the *input* of a second command

```
ls /usr/bin | head -8
```

This pipe

takes the output of the `ls` command and

uses it as the input of the `head` command

The `grep` command searches for a pattern in its input

```
ls /usr/bin | grep dir
```

This pipe
takes the output of the `ls` command and
uses it as the input of the `grep` command,
which here is searching for the pattern “dir”

The `grep` command searches for a pattern in its input, which can be a file

```
grep J /tmp/months.txt
```

`grep` can search in files

`/tmp/months.txt` is a file I created with the names of the months

To move or rename files or directories use the `mv` command

```
mv file1.txt dir_a    # move file.txt to directory dir_a

mv dir_a dir_b        # if dir_b exists:
                      # move dir_a and its contents to dir_b

                      # if dir_b does not exist:
                      # rename dir_a to dir_b

mv file1.txt file2.txt  # rename file1.txt to file2.txt
                        # will overwrite file2.txt
                        # if a file with that name exists
```

Note: it's easy to overwrite existing files with commands like mv

To copy files or directories use `cp` command

```
cp file1.txt file2.txt     # creates a copy of file1.txt
                           # named file2.txt
                           # overwrites file2.txt
                           # if it already exists.

cp file1.txt dir_a         # creates a copy of file1.txt
                           # in directory dir_a
                           # if dir_a exists and is a dir
```

To copy files or directories use `cp` command

```
cp -R dir_a dir_b        # if dir_b exists:
                         # copy dir_a and its contents
                         # into dir_b (will be named dir_a)

                         # if dir_b does not exist:
                         # will create a copy of dir_a and
                         # its contents (named dir_b)
```

To delete files use the `rm` command

```
rm file_a              # deletes file_a


rm -r a_directory      # delete a_directory and any files
                       # it contains.
```

Warning! Files removed by rm will be permanently deleted.

There is no undo for this command and no “Recycle Bin” or “Trash”

Some notes about file naming

(1) Linux filesystems are usually case sensitive

`a_file.txt` and `A_file.txt` are different files

(2) Avoid spaces in file names: they make life difficult

For example, there is a file named `My sequence.fasta` in the `/tmp directory`

Exercise: Write a `cat` command that outputs the contents of this file.

What you can do if you have a space in a filename

(1) Rename the file:

```
mv "/tmp/My sequence.fasta" /tmp/My_sequence.fasta
```

(2) You can quote the file name to use it as is:

```
cat "/tmp/My sequence.fasta"
```

(3) You can “escape" the space using a backslash:

```
cat My\ sequence.fasta
```

Other characters that are problematic for the shell: `;` `&` `(` `)`

To create or delete directories use `mkdir` and `rmdir`

`mkdir`: create a directory

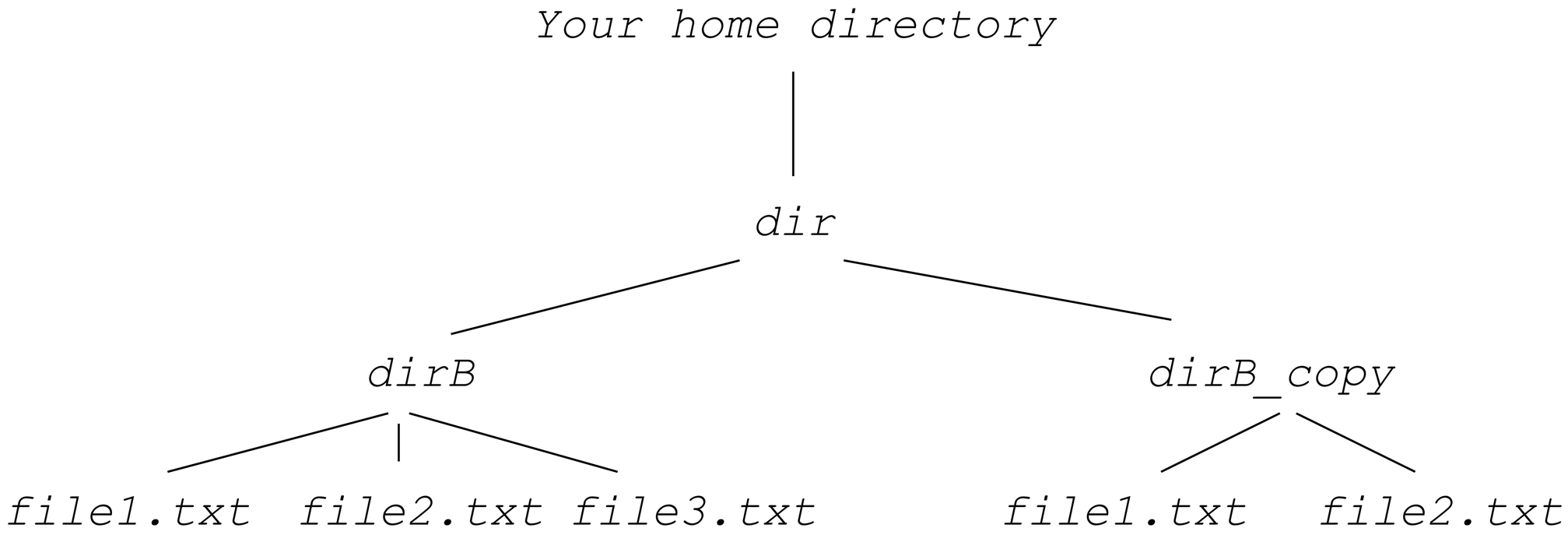
`rmdir`: delete a directory (must be empty)

```
mkdir a_new_directory

rmdir a_new_directory

rm -r a_directory_with_files_in_it
```

Exercise: use shell commands to create directories and files in your home directory with the following organization



Hint: use `touch` to create empty files

Wildcard characters will match any character or characters in file names

```
*       matches any number of any characters
?       matches any one character
```

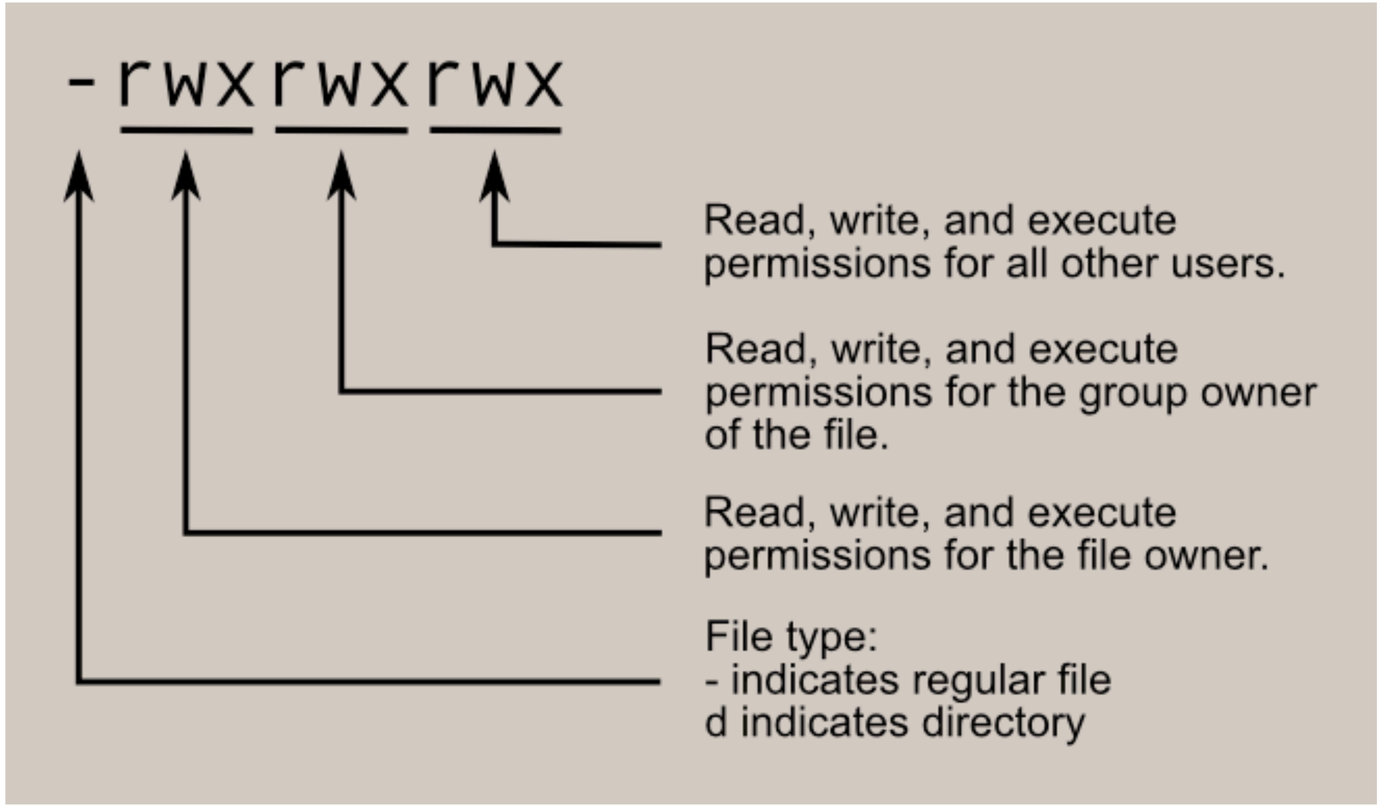
```
Examples:
ls /usr/bin/*dir
ls /usr/bin/*dir*
ls /usr/bin/??dir
ls /usr/bin/??
```

Exercise: capture all of the files in `/usr/bin` that have 2-character names in a file in your home directory named `2_char_commands.txt`

```
*       matches any number of any characters
?       matches any one character
```

Hint: use the output redirection operator to capture output

File permissions in Linux define what can be done with a file and who can do it



Permission types:

Read: required to view or copy a file
Write: required to modify or delete a file
Executable: required to run a command or cd into a directory

Permissions can be different for:

- **Users**
- **Groups** of users
- **All** users

The `chmod` command changes permissions of a file or directory

For example:

```
chmod +x a_file # give a_file executable permissions
```

Shell scripts are just plain text files with shell commands

Any command you can run on the command line you can put in a script
And any command in a shell script you can run on the command line

This has a number of benefits:

(1) It allows you to automate repetitive tasks
(2) It can be used to document what commands you've run
(3) It avoids typos when typing out long command lines

Shell scripts can be run as commands, but have to have executable permissions.

Exercise: let’s create a shell script!

```
# change to your home directory
cd

# use the nano text editor to create and edit a new file
nano my_first_shell_script
```

Enter the following text in nano to create your shell script commands

```
# this file is my first shell script!
# these lines with # characters are comment lines.
# they won't be run.

# let's output a message
echo the current date and time is:

# outputs the date and time
date
```

Type ctrl-x to exit nano and save the file.

Exercise: let’s create a shell script!

```
# change to your home directory
cd

# use the nano text editor to create and edit a new file
nano my_first_shell_script

# try running this shell script
./my_first_shell_script

# whoops: need to give this file executable permissions
chmod +x my_first_shell_script

# try running this shell script again
./my_first_shell_script
```

Linux paths and the PATH

Image credit: James Wheeler (CC BY-NC-SA 2.0)

A path

- The location of a file or directory
- Can be absolute or relative

The PATH

- A list of directories the shell looks in to find commands
- The `which` command tells you if a command is in your PATH
- You can add or remove directories to this list by changing the value of the $PATH variable
- This is why we had to type `./my_first_shell_script` in the previous example: to tell the shell exactly where the command was, since our home directories are not in our PATHs.

The `which` command tells you if a command is in your PATH and if so, where

```
which ls
which which
which cowsay
which sl
which cmatrix
which lsls
```

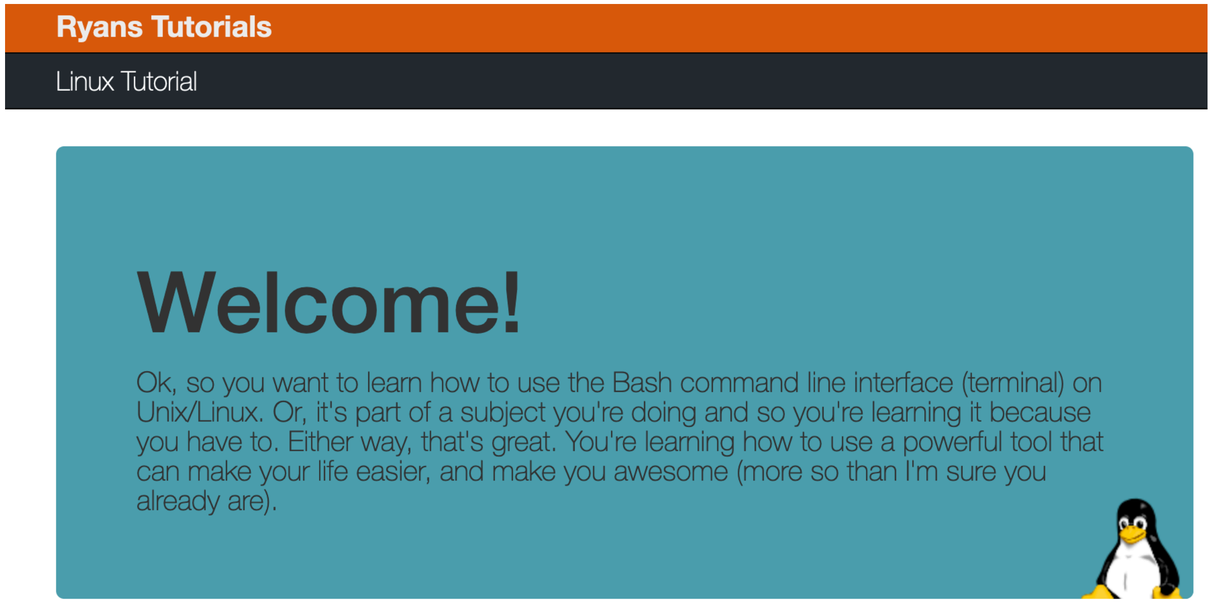
Exercise: create a shell script on your laptop that will run the `ssh` command you use to log into thoth01

On your laptop:

- Use notepad++ or bbedit to create a plain text file with the appropriate command
- Save it with some informative name, like `ssh_thoth` or `ssh_t`
- Give it executable permissions
- Save it somewhere in your PATH
- Use `which` to see if your shell can find it
- Try running it to see if it works

The website linked below is an excellent resource to refer to and to learn more about things we will not have time to cover in class

https://ryanstutorials.net/linuxtutorial/



https://ryanstutorials.net/linuxtutorial/cheatsheet.php