The basics of Linux computing, shell scripting & assorted related topics, starting with the command line

Khaldoun Makhoul
khaldoun@nmr.mgh.harvard.edu

September 29th 2011

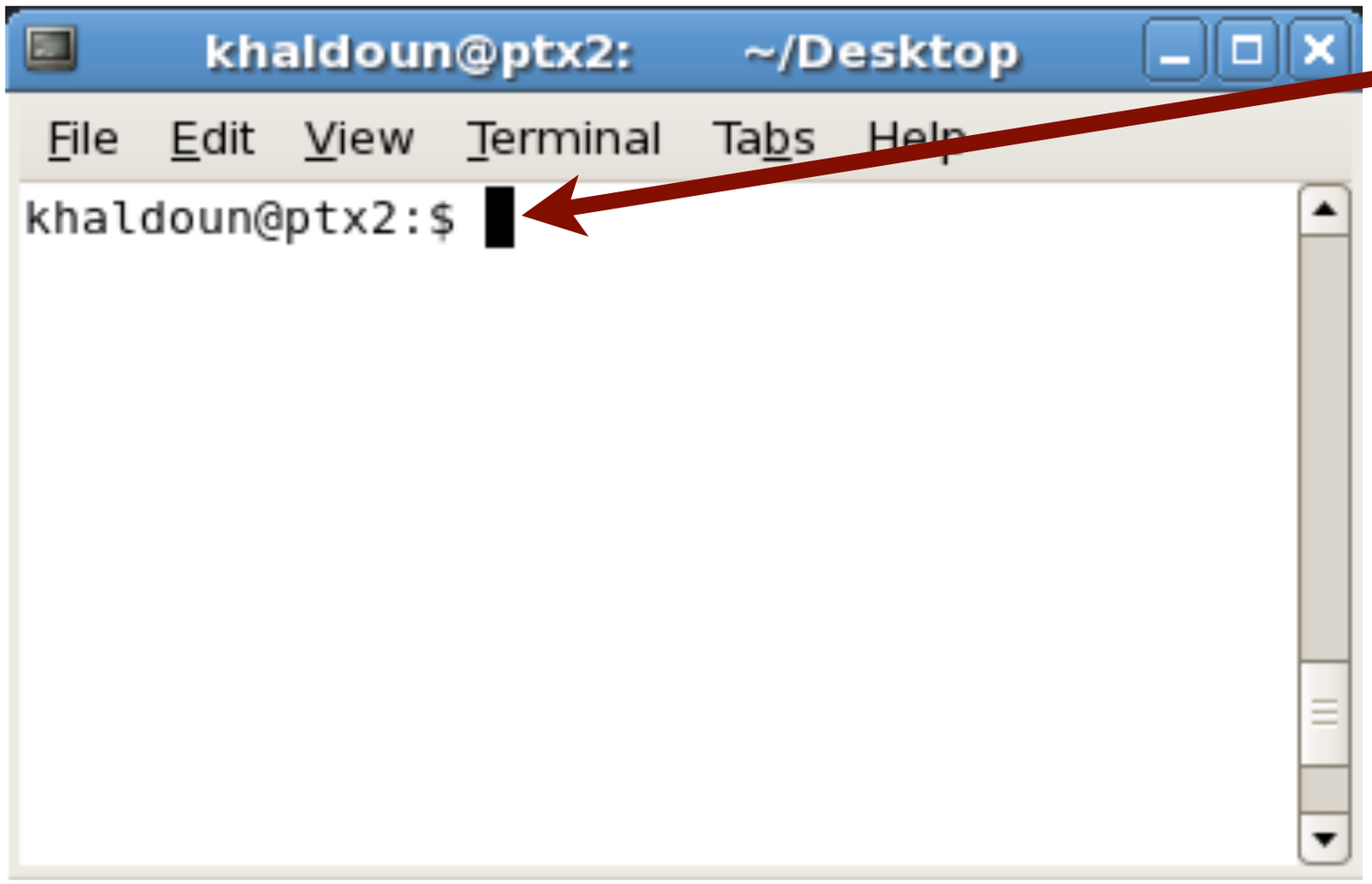
Why.N.How

This talk introduces the audience to the basic use of the Linux command line tools and to basic C shell scripting.

The talk focuses on *using* these tools. I will not go in depth into the inner functioning of Linux, and instead will mostly proceed by example.

This version of the talk is adapted from the one that was given on September 29th and differs from it in a few respects: instead of the live demonstrations of the use of the command line (which was possible during the actual talk), this version contains screenshots and outlines the main steps performed. If there is any confusion or missing information, please contact me at khaldoun@nmr.mgh.harvard.edu

The command line?



The command line is a text interface for giving the computer instructions.

It is there to obey the user's instructions. In that sense, you shouldn't see it as something foreboding and mysterious. It's here to help you do what you want to do!

The only requirement is that you learn the basics of the language needed, and you will then find that it's extraordinarily powerful and useful.

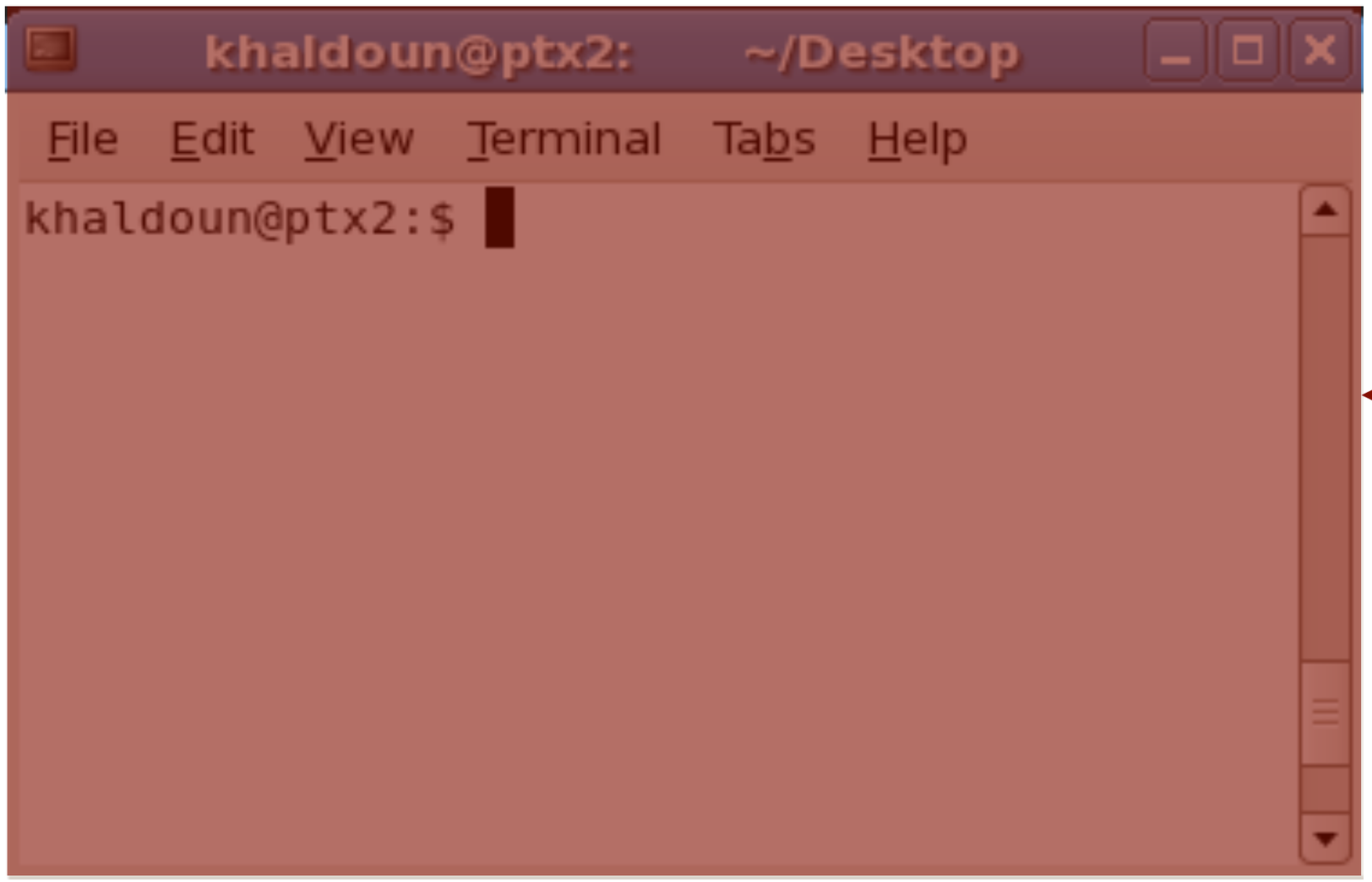
The goal of this talk is to get you to be comfortable at a standard Linux command line, because that's the first step to using the available power and flexibility of this operating system.

If you've never used Linux before, your first question might be "What is the command line?".

The command line is a text interface for giving the computer instructions. The fact that it's text-only is the thing that tends to scare people. The way to overcome this is to learn the basics of the language spoken on the command line. Once you know how to communicate with it, it's no longer daunting.

(I mean "language" in a rather informal sense here, although naturally this also refers to an actual *programming* language, that is, the language of the shell that interprets your commands... for more on that, see part 2 of this talk!)

The command line in a terminal



OK, but the command line isn't just floating on your screen.

It's displayed inside a terminal window

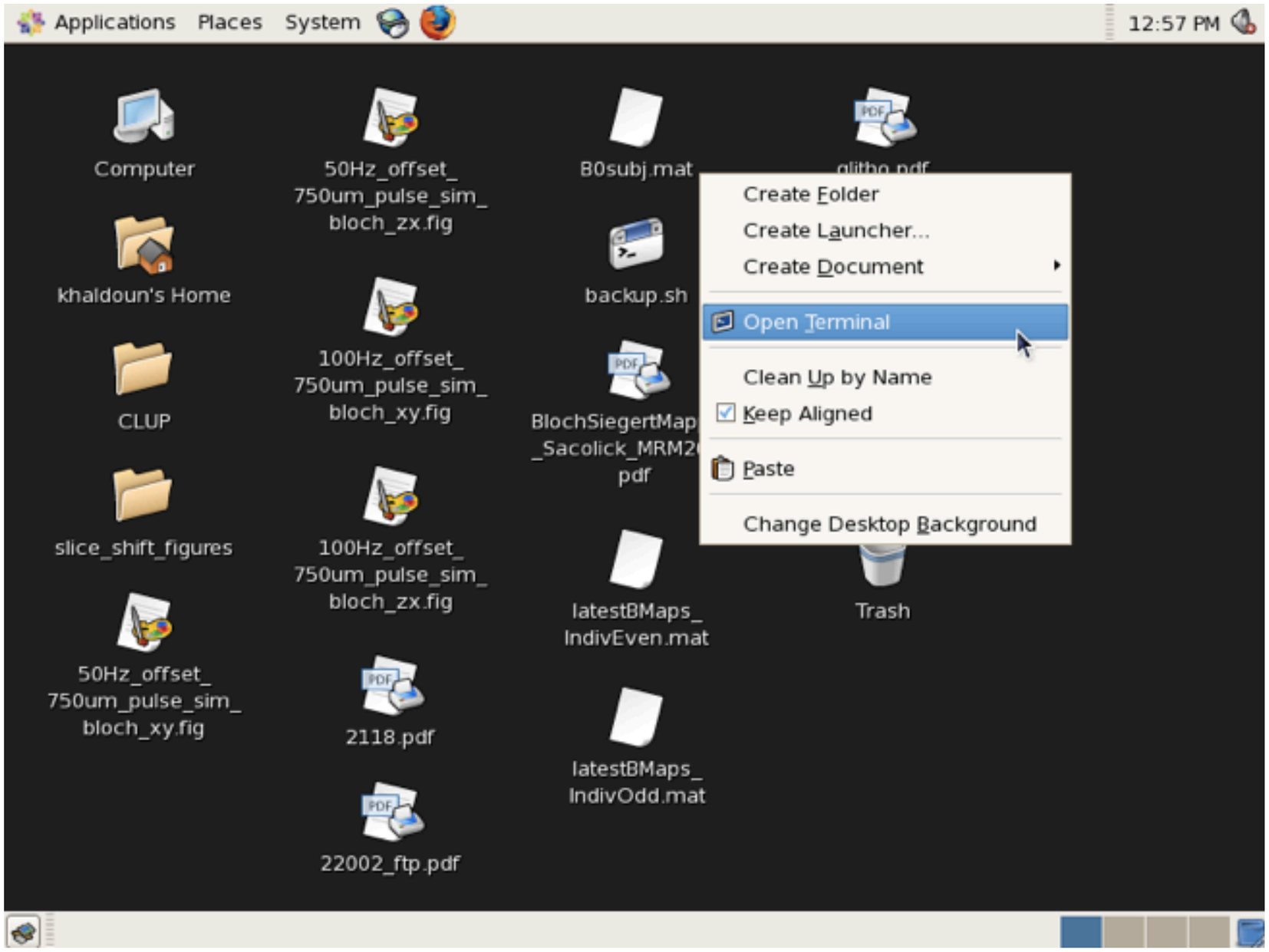
Next question you might have: how do I get to this command line?

For all intents and purposes, you'll always be entering commands at the command line inside a terminal program. That's the program that you see in the screenshot above. The menu bar, the scroll bar, the minimize/maximize/close widgets, those are all part of the terminal program, which has a dual purpose:

-- allowing you to enter commands
-- giving you text feedback

In that sense, it's the space in which a dialogue between you and the command line takes place.

Starting a Terminal from CentOS



On any remotely mainstream Linux installation, the post-login experience looks very similar to Win/Mac... so how do we get to the part that's fun?

Right-click on the desktop background and choose "Open Terminal"

On CentOS, you can always right-click on the desktop background after you log in and choose "Open Terminal".

There are many many many other ways to do this, but this is the simplest and most relevant for people at Martinos.

Why the command line?

My desktop (folder) seen from the command line

```
khaldoun@ptx2:~/Desktop
File  Edit  View  Terminal  Tabs  Help
khaldoun@ptx2:$ cd ~/Desktop/
khaldoun@ptx2:$ ls
100Hz_offset_750um_pulse_sim_bloch_xy.fig
100Hz_offset_750um_pulse_sim_bloch_zx.fig
2118.pdf
22002_ftp.pdf
50Hz_offset_750um_pulse_sim_bloch_xy.fig
50Hz_offset_750um_pulse_sim_bloch_zx.fig
B0subj.mat
backup.sh
backup.sh~
BlochSiegertMapping_Sacolick_MRM2010.pdf
CLUP/
latestBMaps_IndivEven.mat
latestBMaps_IndivOdd.mat
qlitho.pdf
Ramsey_Phys1955.pdf
slice_shift_figures/
TXSwitch_small-NoTrap_w2.brd*
khaldoun@ptx2:$
```

My desktop seen from a randomly chosen "modern" GUI (Graphical User Interface)

Next possible question: why the command line?

This slide could be seen as the argument "against": a modern graphical user interface (GUI) looks more sophisticated, possibly easier to use, and just plain prettier. If you give a user who's never used a computer one of these two interfaces, it's probable that they can learn to use a GUI more easily than they can learn a text-only interface.

Since it's so plainly ugly, why use it?

Old-school Command Line Interface

- Power
- Flexibility
- Speed
- Scriptability

But if you are using your computer to get work done, you're not necessarily concerned much with aesthetics, or with ease of use*. What you care about is the following: **power, flexibility, speed, and automation (scriptability), and the like**. Those things will make your work go faster, and will allow you to be more successful. These are **precisely** the things for which a command-line interface is excellent.

* (up to a certain extent, and not to imply that things should be hard to use just for the sake of difficulty)

Outline

This talk will proceed by practical example. I will expand on relevant concepts as they come up.

Some basic Linux commands & structure of statements

1. Intro: `ls`, command syntax, etc
2. Making `rm` safer
3. Grab-bag of basic commands & info
4. `grep` and `find`

} Basic "vocabulary & syntax"

Intro to scripting

5. A sample script
6. Output redirection & pipes
7. Input parameters
8. Looping
9. Conditional statements

} More complex "grammar & language"

The talk is divided into two sections. In the first part, we will to a few examples to demonstrate basic commands. This is to get you familiar with doing a few simple tasks, but also to start getting an idea of the basic structure of commands. In the second section, we will learn a few basics of scripting.

Just as when you're learning a foreign language, it's good to start with some basic vocabulary and the rudiments of syntax. Then once you have a "feel" for the language, we can do some more complex grammar.

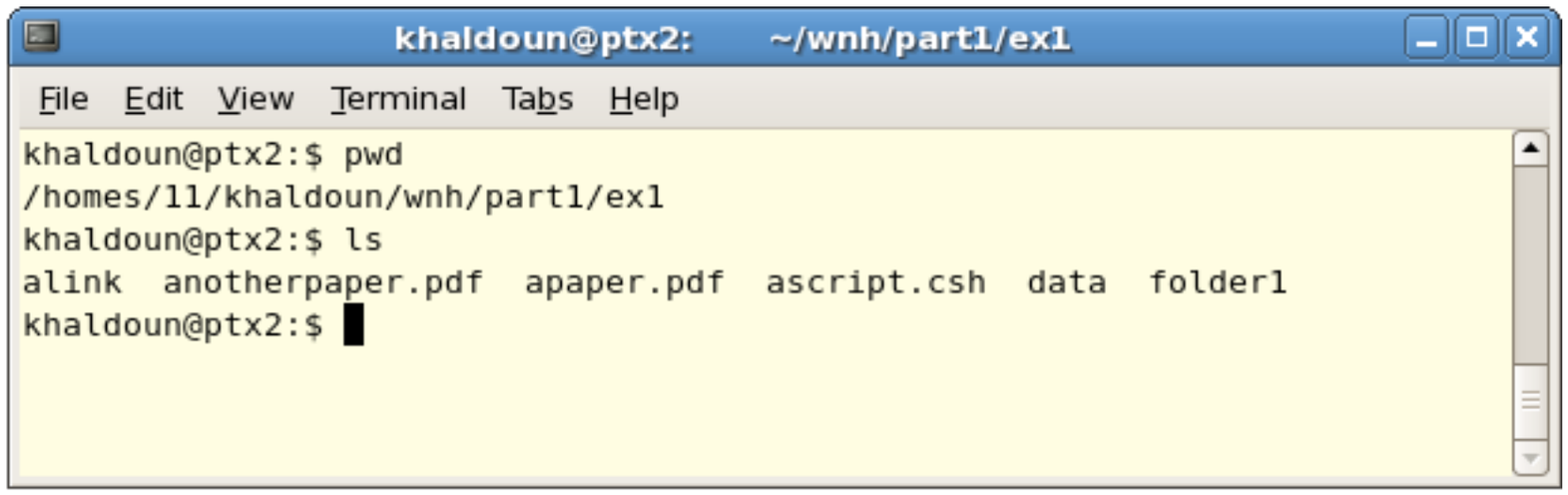
Example 1: making `ls` more useful

- `ls` is the command to list files in a given directory (folder)
- By default, the text-only listing is a little too plain
- We'll change that, and learn a few basic commands along the way
- Commands/programs/files introduced:
  - **`pwd`** (tell me the current directory)
  - **`ls`** (list files)
  - **`man`** (diplay manual pages)
  - **`alias`** (replace typed command by another)

Listing files is not the most exciting thing in the world, but it will give us the opportunity to learn a lot about using any command in Linux.

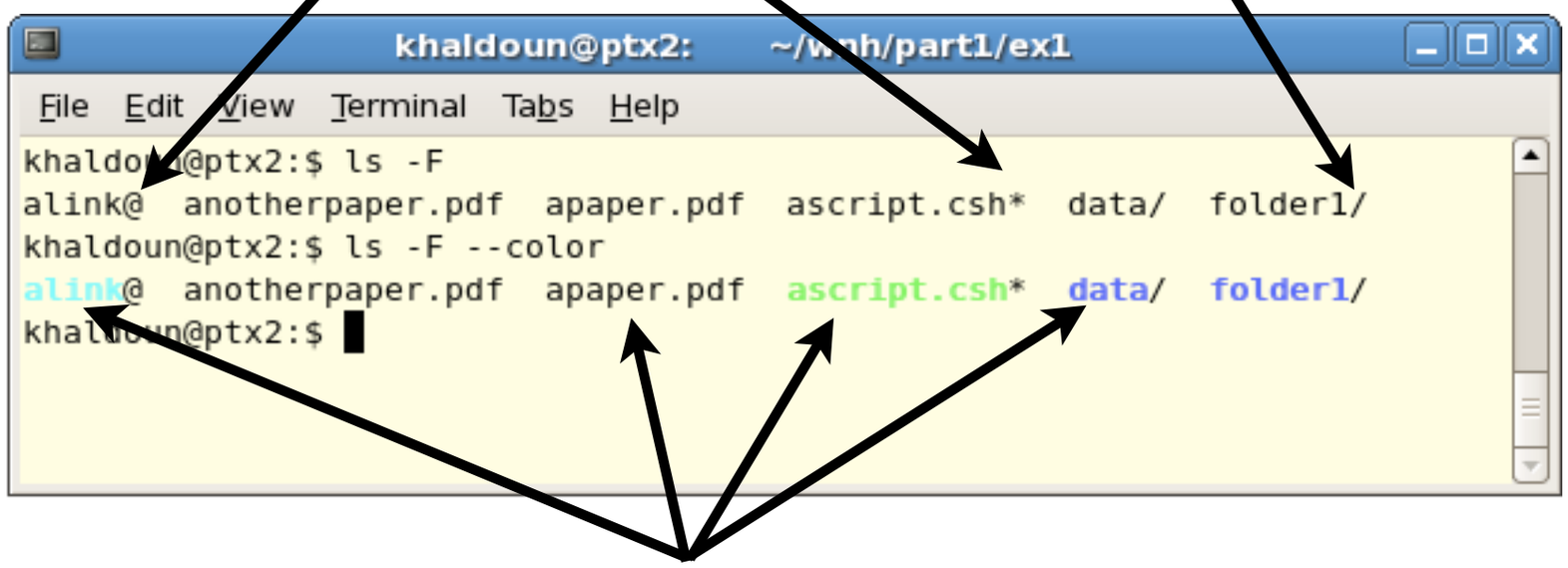
This example will also familiarize you with various useful commands, listed here.

Example 1a: getting some color



First, I type `pwd` to find out what folder I'm currently in. Then I type `ls` to list the contents of that folder.

`-F` appends `@` for links, `*` for executables, and `/` for folders



`--color` shows different file types in different colors

As an example of the flexibility available, let's enable some options to distinguish between different file types:

`-F`

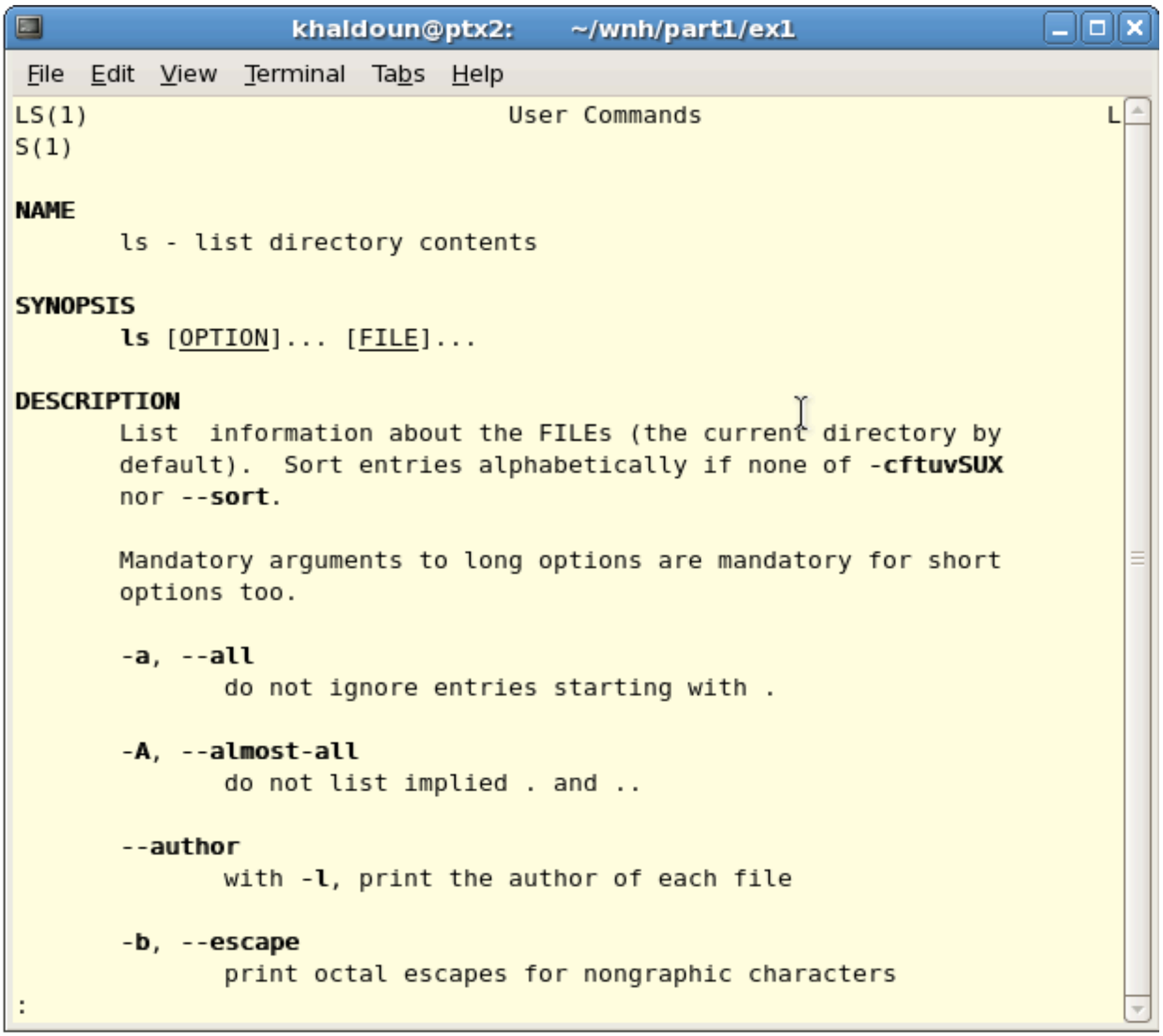
`--color`

(Note that you can combine them)

Note: always press ENTER or RETURN to execute the command(s) you've just typed. ENTER is not carriage return (next line)! Think before you validate :)

Note: the color settings I have may not be the ones that are used on your system.

Example 1b: manual page for `ls`



But I remember there was another file in that folder.... perhaps it's hidden. How do I check?

Your first stop when learning about a particular command is the manual (or "man" page). Type `man ls` to view the manual page for the `ls` command.

The manual pops up and takes up the whole Terminal screen (you can type `q` to quit & go back to the command line).

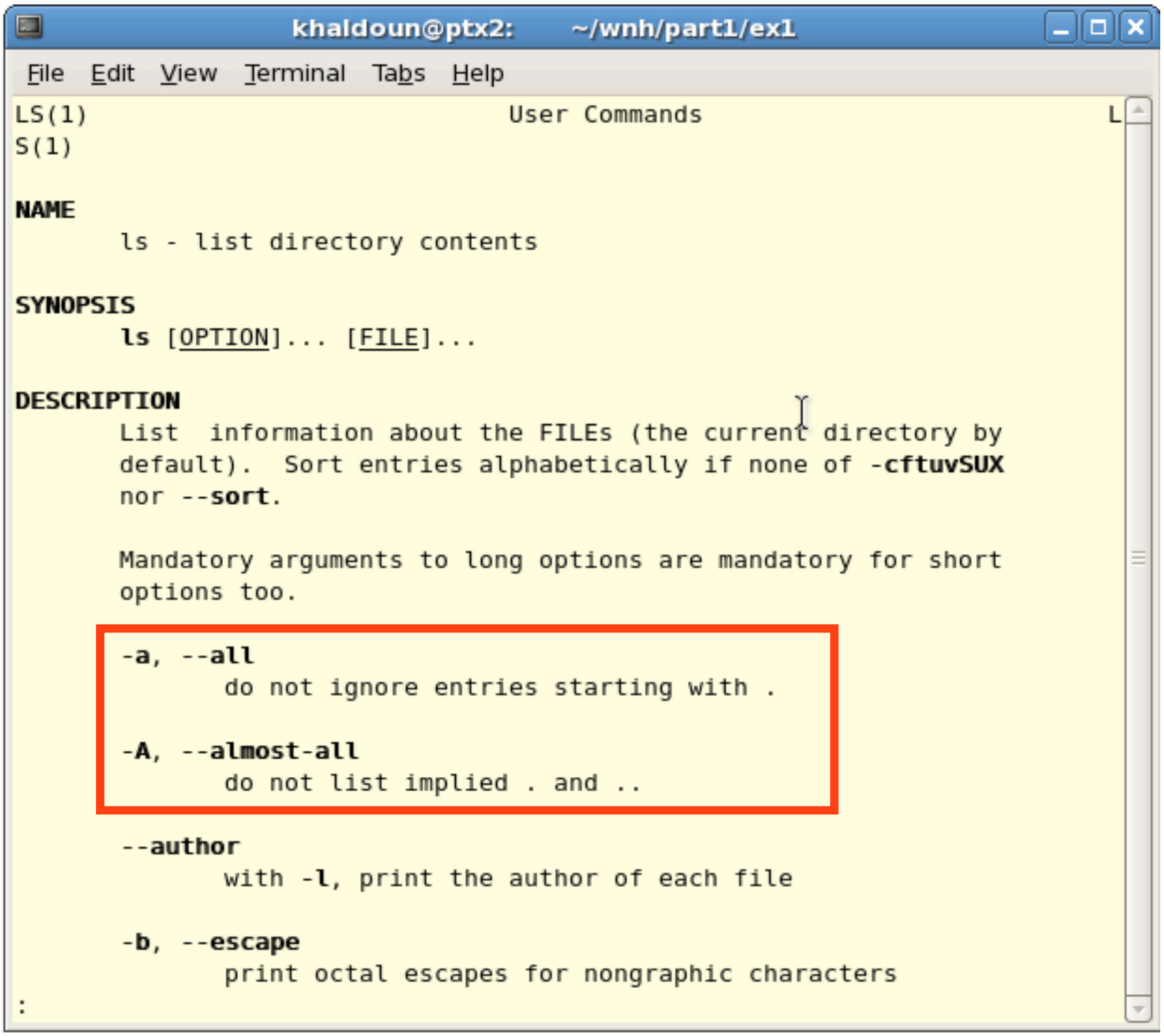
Now let's see if we can glean something about `ls` by reading its manual entry. Type `man ls` and hit ENTER. `man` is a terrifically useful resource. Anytime that you are having trouble with a command, or that you're not sure how a particular command works, your first stop is to check whether it has an entry in `man` (not all commands do). If it does, it's often the best way to learn how a command works.

Get used to using the `man` pages!! They are very useful, and should be your first stop, followed quickly by a web search if that doesn't turn up enough information.

Note: to scroll by a full page in `man`, hit SPACE; to scroll by one line, use the up and down arrow keys. To scroll backwards by a full page, hit the letter `b`. To quit and return to the command line, type `q`. To search for a phrase, type `/`, then type the query, then type ENTER. While in search mode, hit `n` to go to the next match, and `p` to go to the previous match.

You can learn more about man by typing `man man` and hitting ENTER.... but sadly, it's not obvious how to navigate from the `man` entry for `man`, which is why I include it here.

Example 1c: `ls` option to show all



The options `--all` and `--almost-all` list all files in the folder, including hidden files (on Linux, any file that begins with a period is hidden).

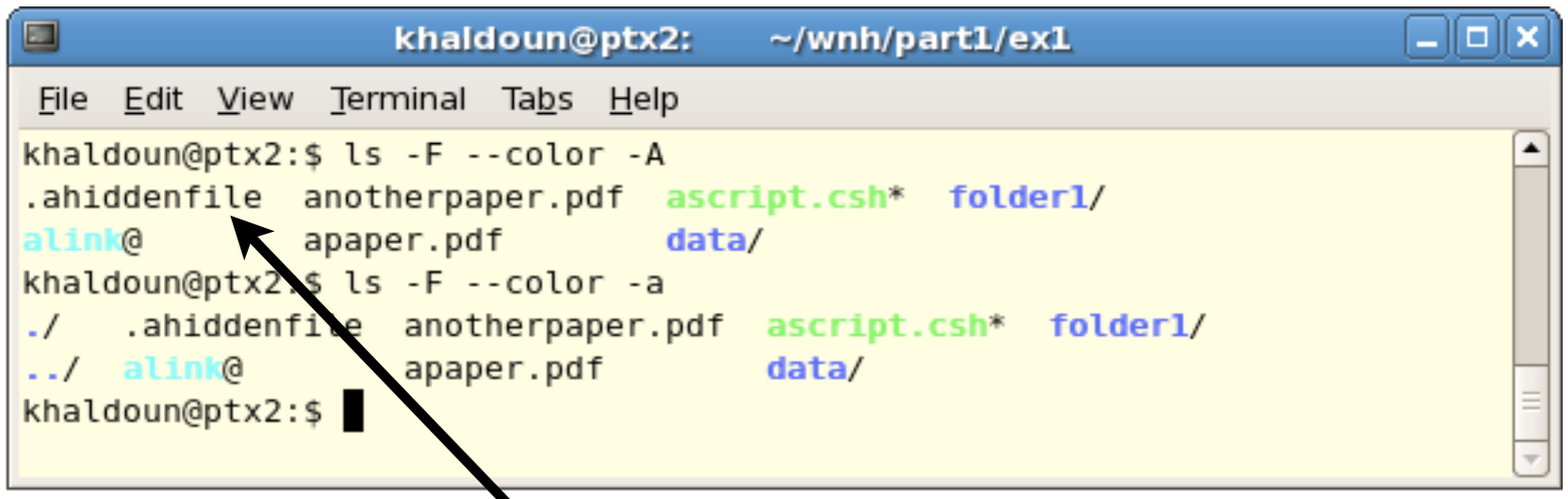
This is what we're looking for!

We can use the abbreviated versions of those options, `-a` and `-A`

Note the structure of a `man` entry:
- -Name
- -Synopsis (usage)
- -Description
- -Options

Note: to scroll by a full page in `man`, hit SPACE; to scroll by one line, use the up and down arrow keys. To scroll backwards by a full page, hit the letter `b`. To quit and return to the command line, type `q`. To search for a phrase, type `/`, then type the query, then type ENTER. While in search mode, hit `n` to go to the next match, and `p` to go to the previous match.

Example 1d: listing hidden files



There's that file!

The option `--all` (`-a`) lists two more entries than the option `--almost-all` (`-A`):

- `./` is the current directory (`ex1/`)
- `../` is the parent directory (`part1/`)

They're just shortcuts for you to refer to these directories when you need to.

Example 1e: more command options

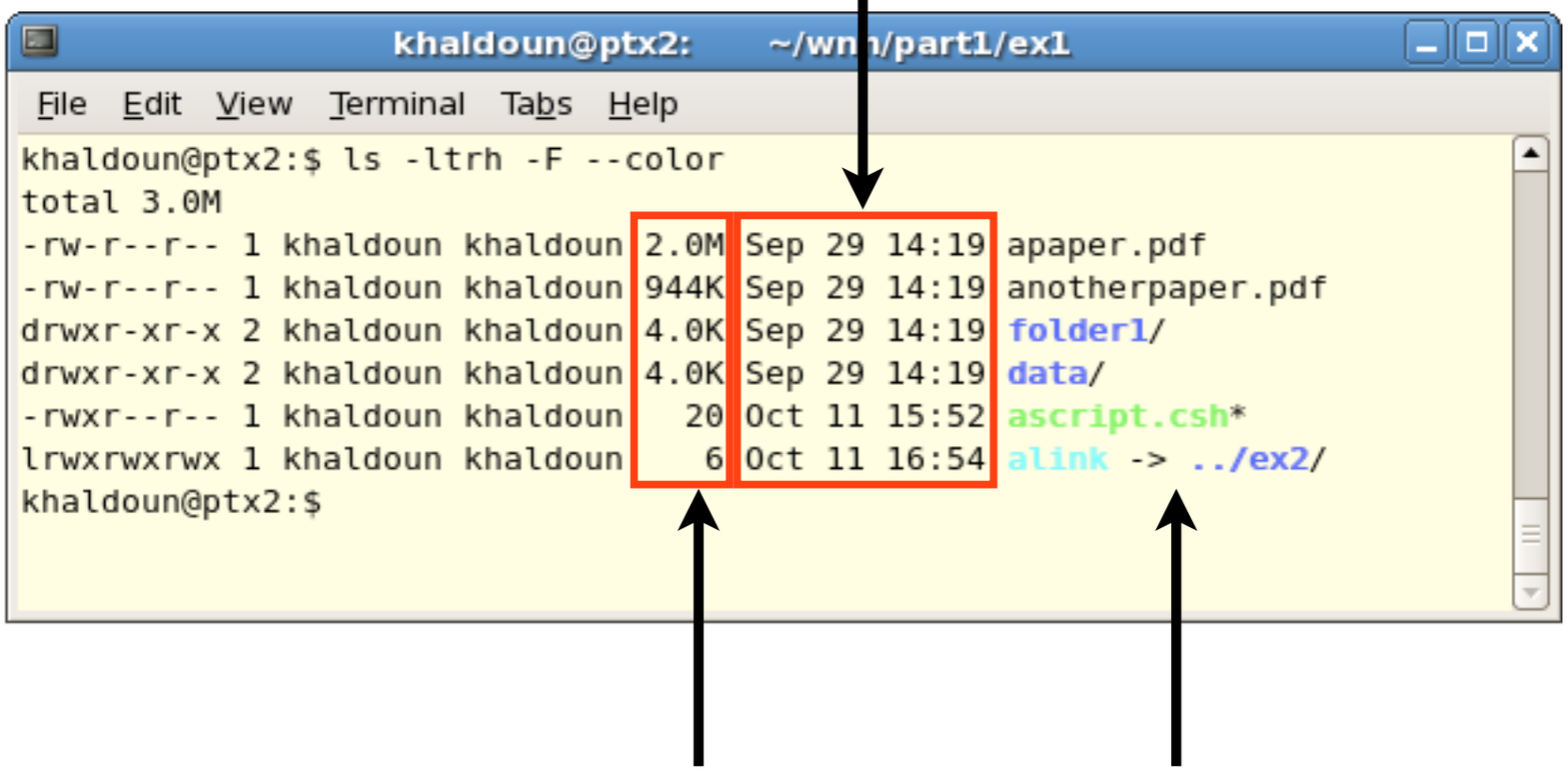
Almost all commands include options you can invoke if need be. The syntax is usually `command -option`. Taking `ls` as an example (just a few among many!):

- **`ls`** `-l` (list in long format)
- **`ls`** `-a` (list all files including hidden)
- **`ls`** `-t` (list and sort by time)
- **`ls`** `-r` (list and reverse sort order)
- Combinations possible: **`ls`** `-latr` (list all files in long format in reverse order of recently modified)

Check the manual page for more!

Example 1f: testing command options

The option `-t` lists in chronological order, with oldest at the bottom.
The option `-r` reverses the sort order (now newest at the bottom).



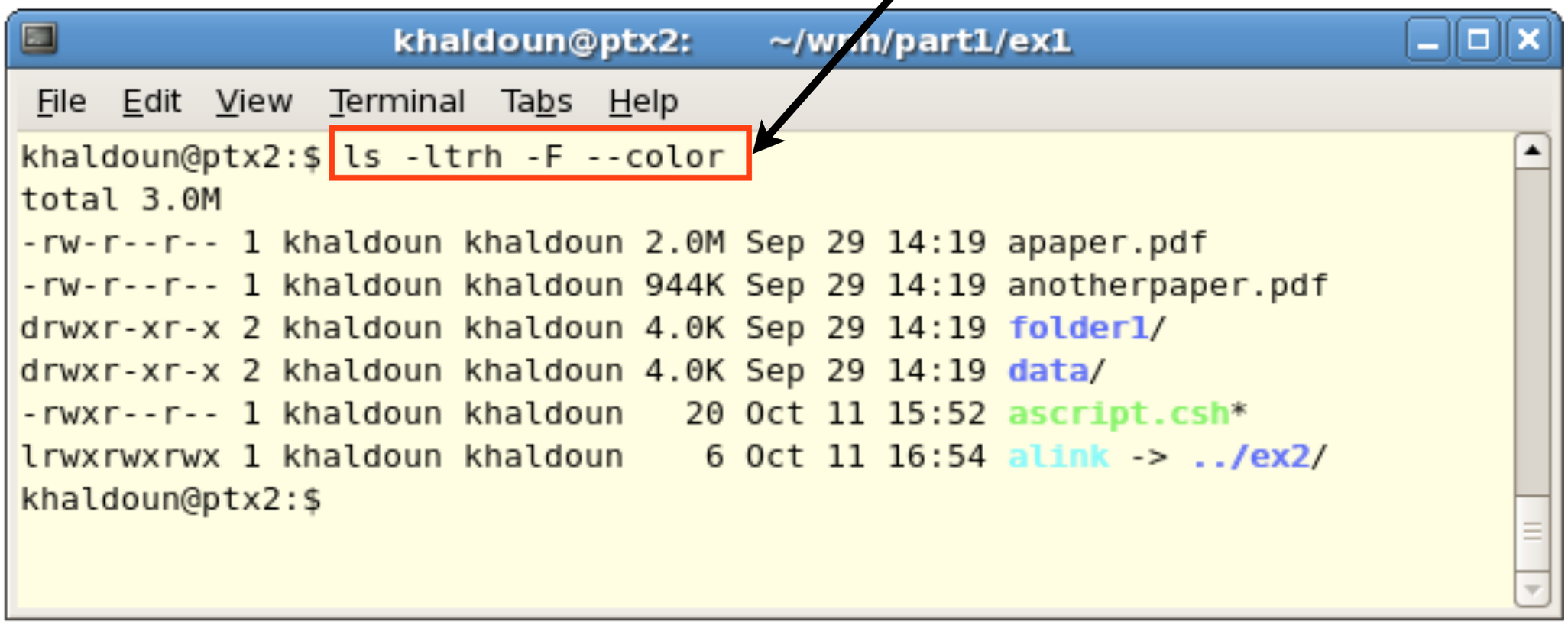
The option `-l` lists in long format, showing file permissions, owner, file fize, creation date, etc.

The option `-h` displays file sizes in "human-readable" format (i.e. with the G, M, K abbreviations for gigabyte, megabyte, etc)

Note that with the `-l` option, the output shows you where the link points, in this case to the folder `ex2/` in the parent directory, or `../ex2/`

Example 1g: using aliases

This command is getting much too long!!



I want the following options to ALWAYS be active:

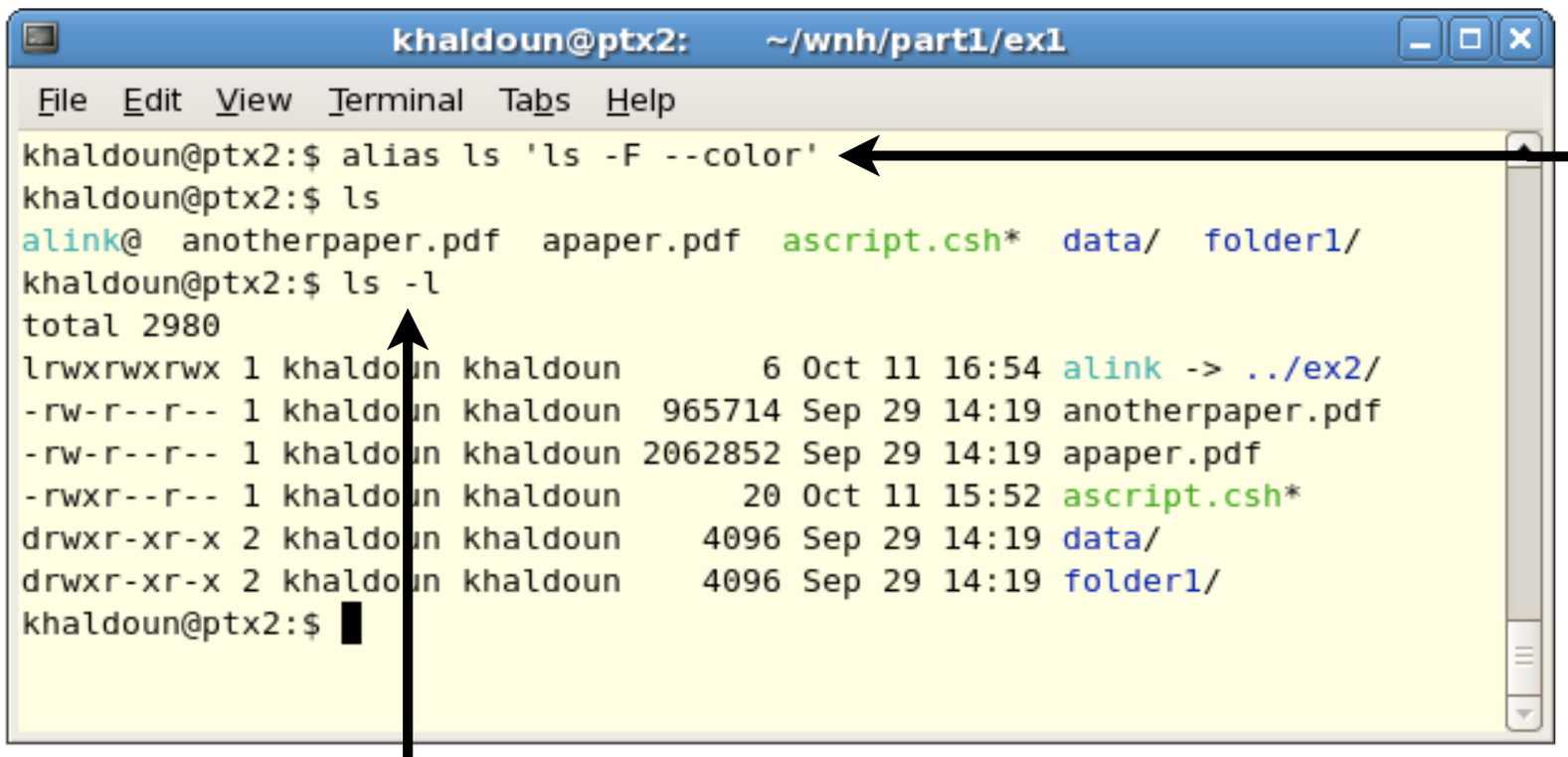
`-F`

`--color`

`-h`

(`-h` is debatable, and your preferences may vary)

Example 1h: one-time aliases



`alias` allows me to tell the computer: when I type some command `xyz123`, interpret it as some other command `abc456`.

In this case, I want to tell it that whenever I type `ls`, it should interpret it as `ls -F --color`.

Note that the alias we created still works even if you add more options. This line is read as `ls -F --color -l`.

Any aliases that you create this way will no longer be applied when you close that session. We'll see later how to make these more permanent.

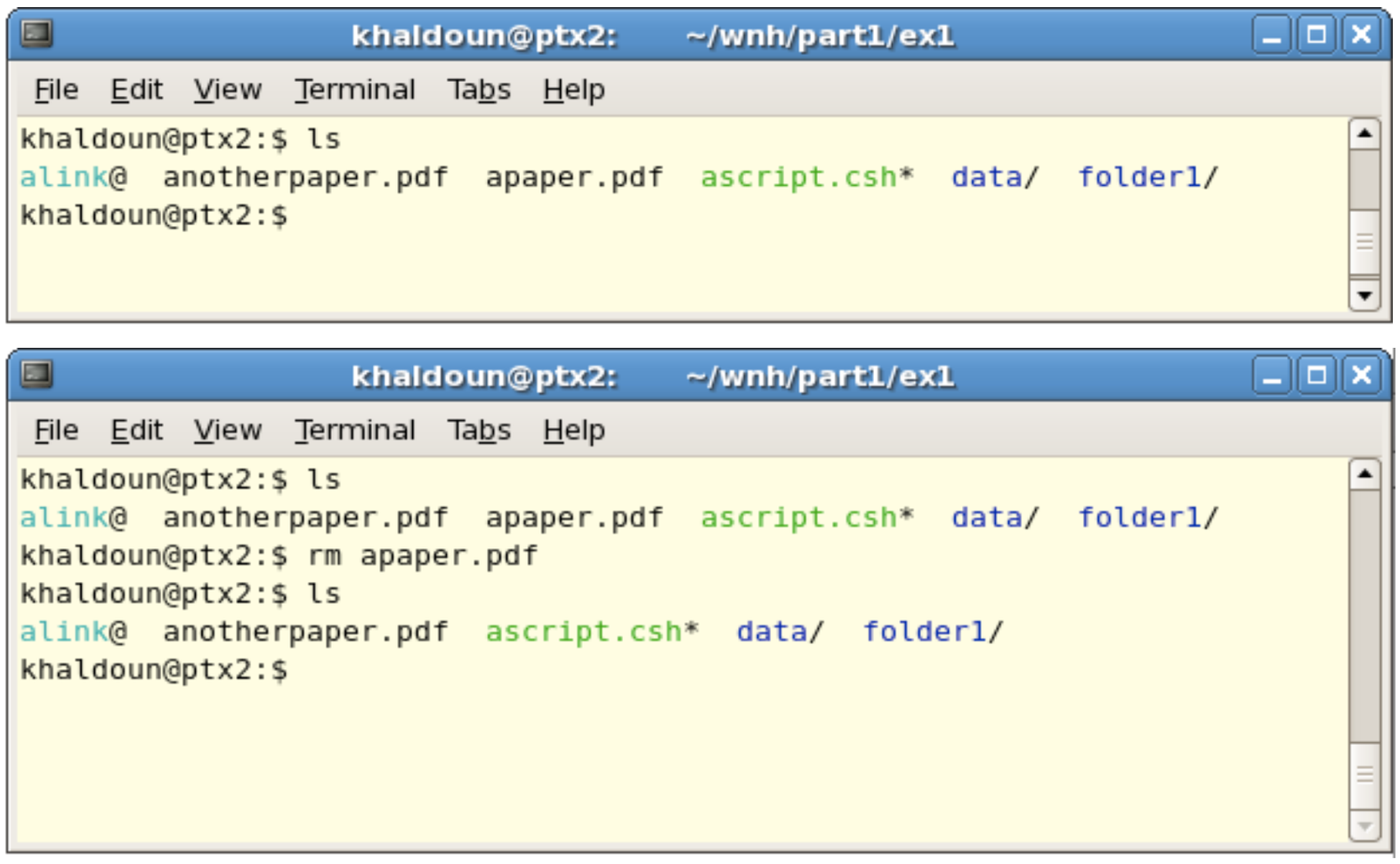
Example 2: making `rm` safer

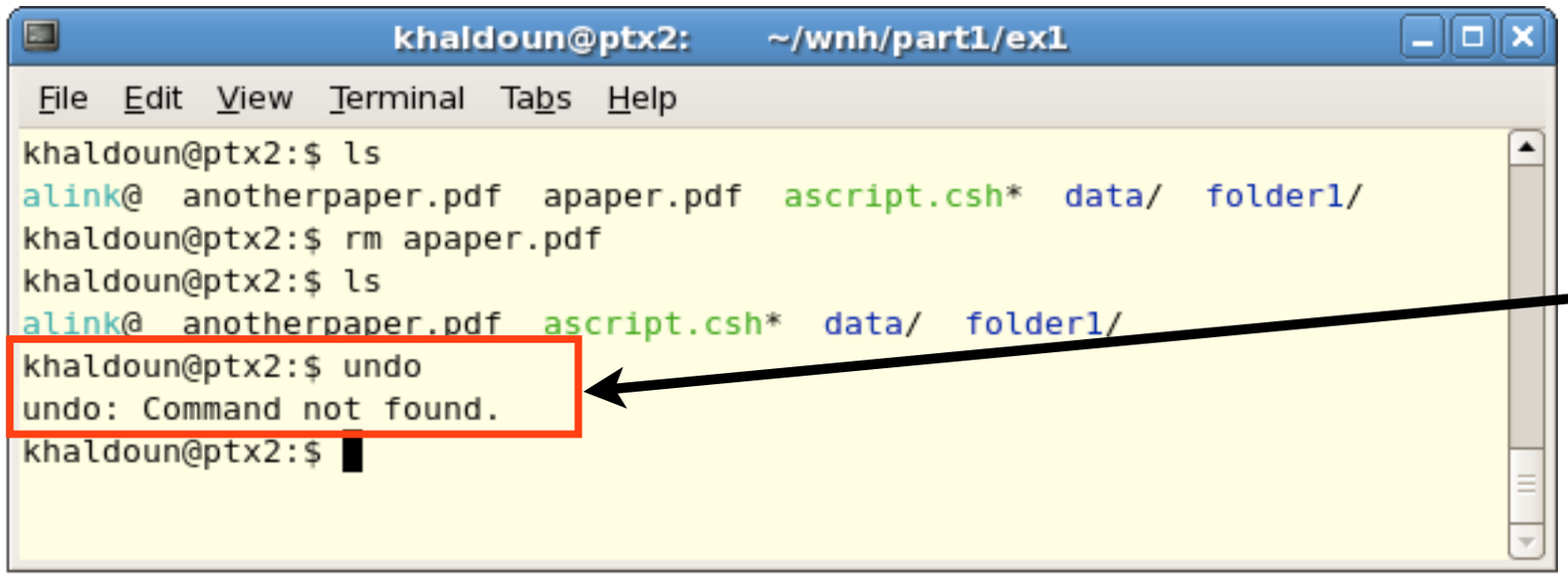
- `rm` is the command to delete files and folders
- By default, the command deletes without asking for user confirmation
- We'll change that so that we don't accidentally delete anything
- Commands/programs/files introduced:
  - **`rm`** (delete files and folders)
  - **`pico`** (text editor)
  - `.cshrc` (c shell config file)

Let's use some of the things we've learned here so far, and apply them to another command.

The command line grants the user a great deal of power, which sometimes means it also grants more power to break things. It's important to feel comfortable when learning to use the command line, so it's best to minimize the chance that anything will go wrong.

Example 2a: default `rm` behavior





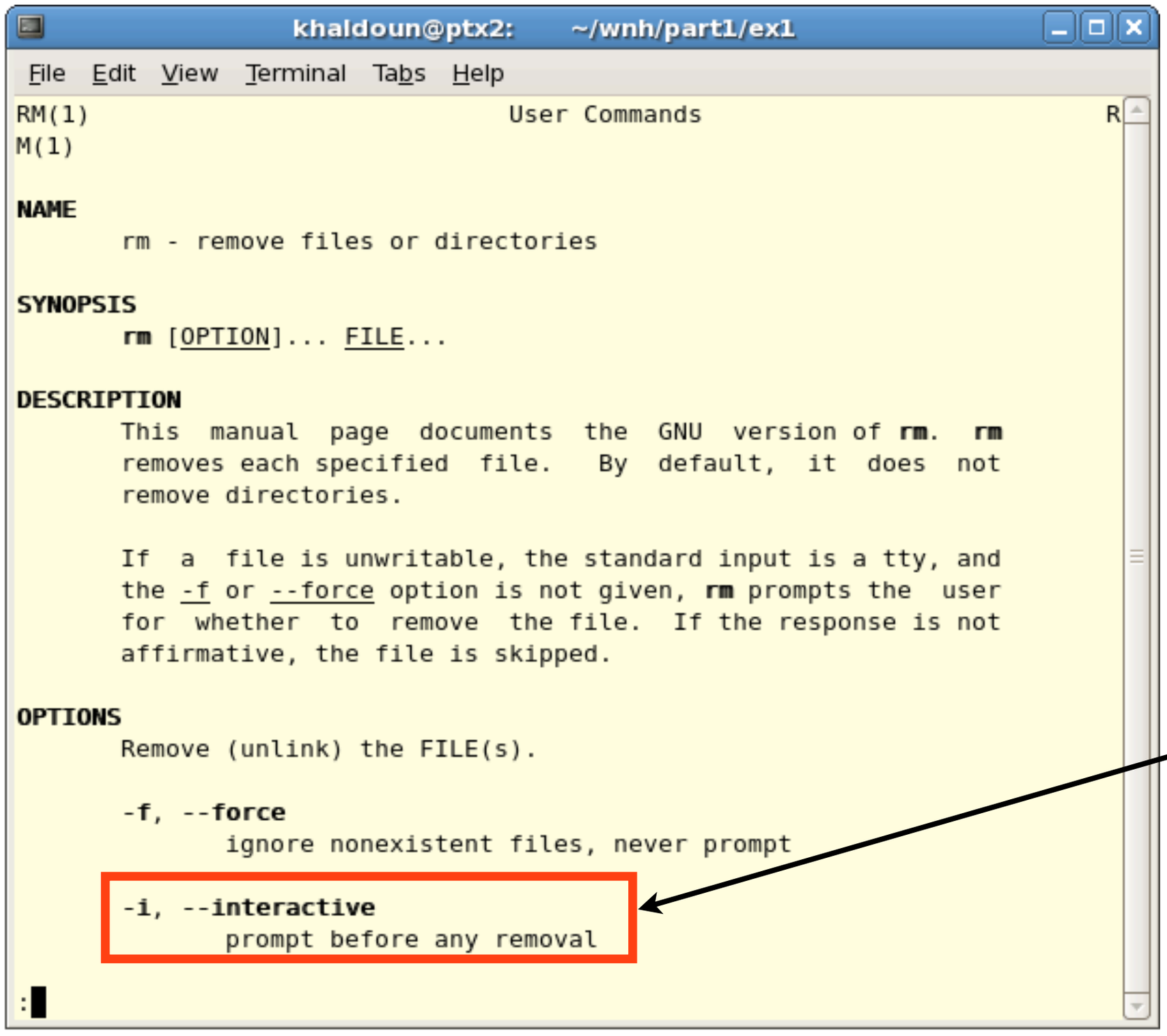
Type `ls` to list the contents of the current directory (folder)

Type `rm apaper.pdf` to delete that file forever. Note that the next `ls` shows that `apaper.pdf` is gone.

No undo here!

`rm` (short for “remove”) is the command to delete a file or files (possibly including folders depending on command line options – see later slides). It is a problematic command at first because it acts right away without confirmation (“Are you sure you want to...”) and without a recycle bin. It’s best to correct this behavior so as to make it safer by adding a confirmation. At the very least, you’ll feel more comfortable knowing that you’re much less likely to delete anything accidentally.

Example 2b: `rm` manual page



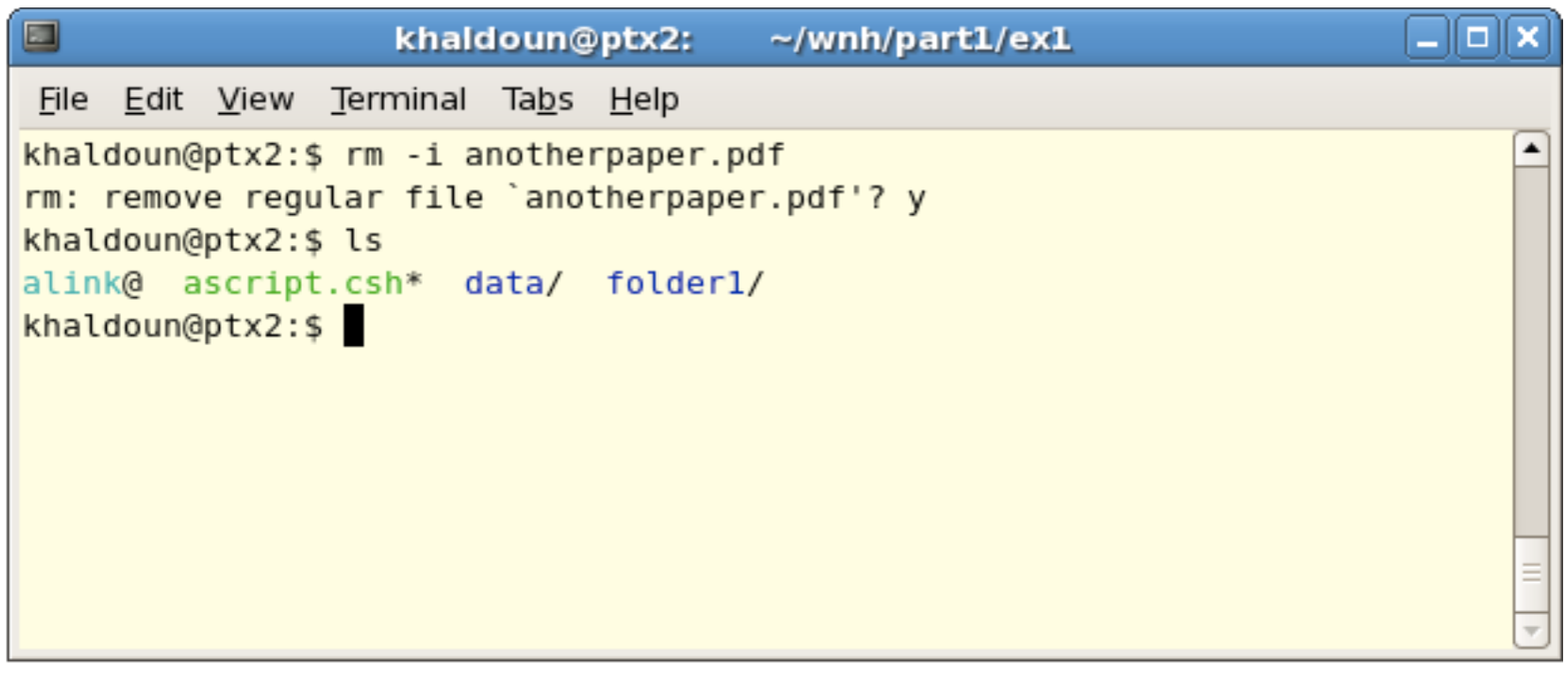
Type `man rm` and hit ENTER to show the manual entry for `rm`

What we're looking for!

Now let's see if we can glean something about `rm` by reading its manual entry. Type `man rm` and hit ENTER.

Note: to scroll by a full page in `man`, hit SPACE; to scroll by one line, use the up and down arrow keys. To scroll backwards by a full page, hit the letter `b`. To quit and return to the command line, type `q`. To search for a phrase, type `/`, then type the query, then type ENTER. While in search mode, hit `n` to go to the next match, and `p` to go to the previous match.

Example 2c: `rm` confirmation dialog



Now if we type `rm -i anotherpaper.pdf`, the system asks to confirm first.

Any answer other than "`y`" will be interpreted as "no", including just typing ENTER

Note: the confirmation "dialog" will specify the type of file that you're trying to delete. In this case, I am deleting a regular file (not a directory, not a link, etc), hence the message you see here.

Example 2d: alias for `rm`

Of course having to remember to type "`rm -i`" each time isn't exactly much safer.

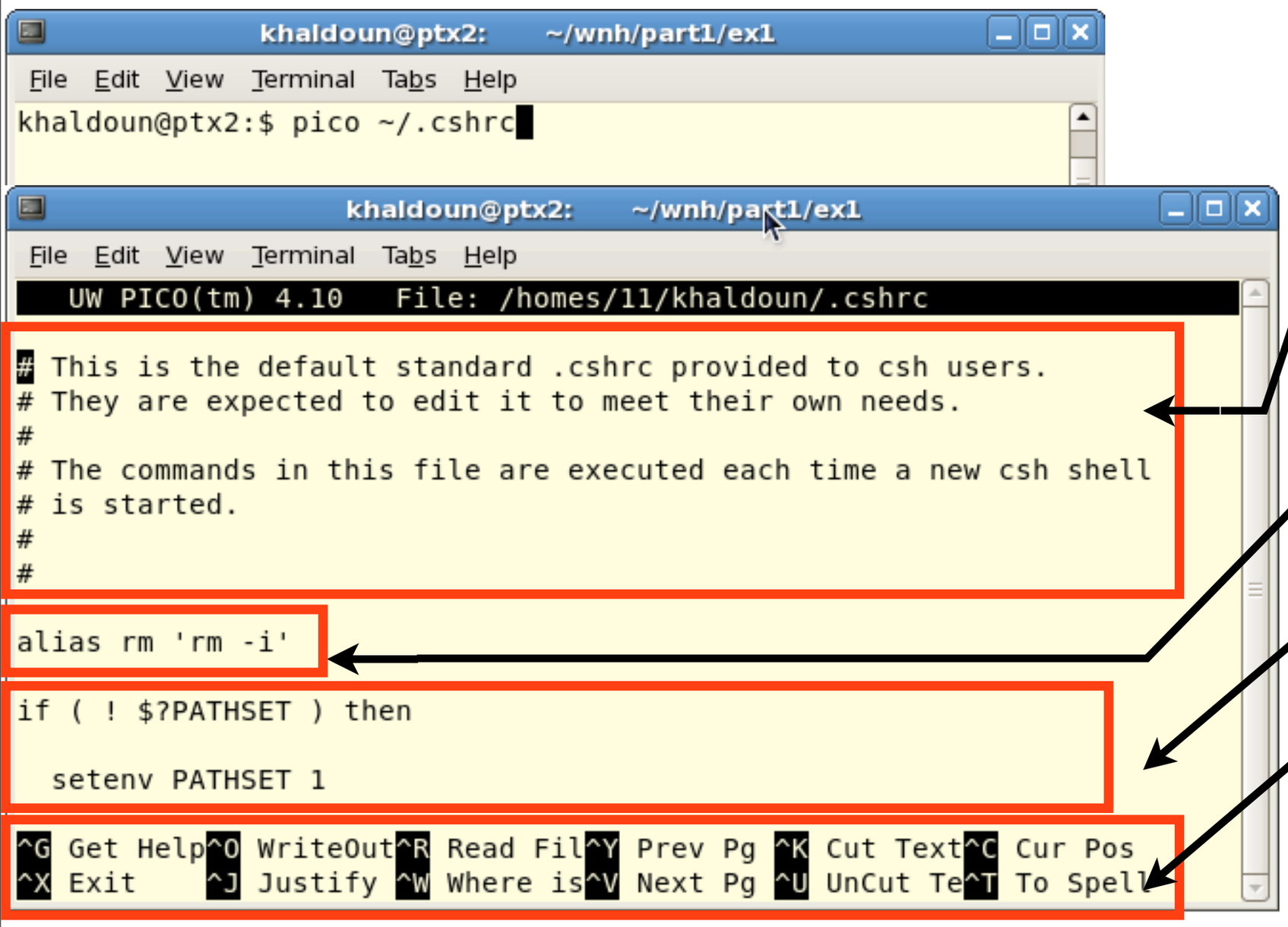
But now you know that this is where `alias` comes in.

We tell the system that `rm` is now an alias for `rm -i`, and verify that this in fact works as expected.

There are two things going on here. We could make an alias that allows you to type "`del`" of "`safe_rm`" or anything you like to mean "`rm -i`". But if safety is what you're after, you can simply make it so that you alias "`rm`" itself to mean "`rm -i`".

Example 2e: alias for `rm` in config file

Finally, you should note that any alias you use is only in use during your current log in session. As soon as you close that terminal, all the aliases you created will be gone. To make this change permanent, you can include it as a line in your `.cshrc` file, which is a configuration file that is read-in every time you open a new c shell. Here I use the text editor pico to perform this task.



Comments (indicated by the initial "#"): these lines are not interpreted.

The line we're adding

Previously existing lines: we can leave these alone

Built-in instructions for pico: ^X means Control-X

We want to modify the file `.cshrc` (pronounced, "dot-C-S-H-R-C". The period at the beginning is the first character in the file name, and is not optional). This file is located in your home directory (a sort of "My Documents" for Linux). The character "~" is an alias for your home directory, whatever its actual location in the filesystem is. The slash, /, is the separator between folders in a hierarchy, or between the folder and the file at the end of the file path.

You will need to use a text editor at some point in your work, and you might as well get used to it asap. This is not the same as a word processor in that a text editor always manipulates plain text files (no fonts, no page layouts, etc... just text). pico is a good first choice because it comes with built-in instructions on how to use it as soon as you launch it (see the bottow of your terminal window). To save the file after you've modified it, type ^O (Control-O). To exit, type ^X (Control-X).

Example 3: Other basic commands

- **`mkdir`** `dirname` (make directory, i.e. a folder)
- **`cd`** `dirname` (change directory, i.e. go to that folder)
- **`cp`** `source destination` (copy files/dirs)
- **`mv`** `source destination` (move or rename)
- **`less`**/**`more`** `textfilename` (display contents of file)
- **`cat`** `textfilename` (concatenate contents of text file)

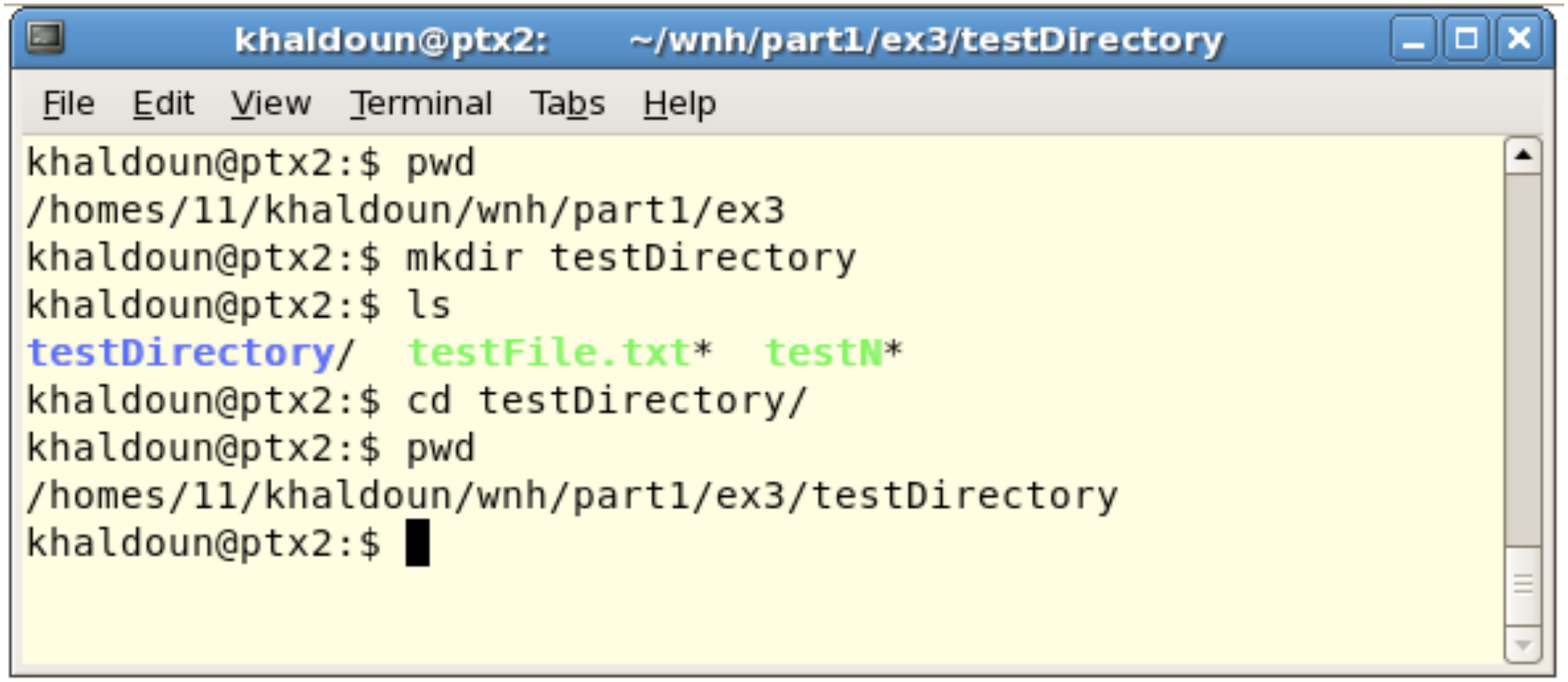
Syntax is almost always one of the following:

- **`command`**
- **`command`** *`options`*
- **`command`** `arguments`
- **`command`** *`options`* `arguments`

Unless you spend your time on the computer deleting files, you'll want to know a few more commands to get started. These are the basics of the basics. Your fingers will probably end up typing these out automatically within a few days of Linux use.

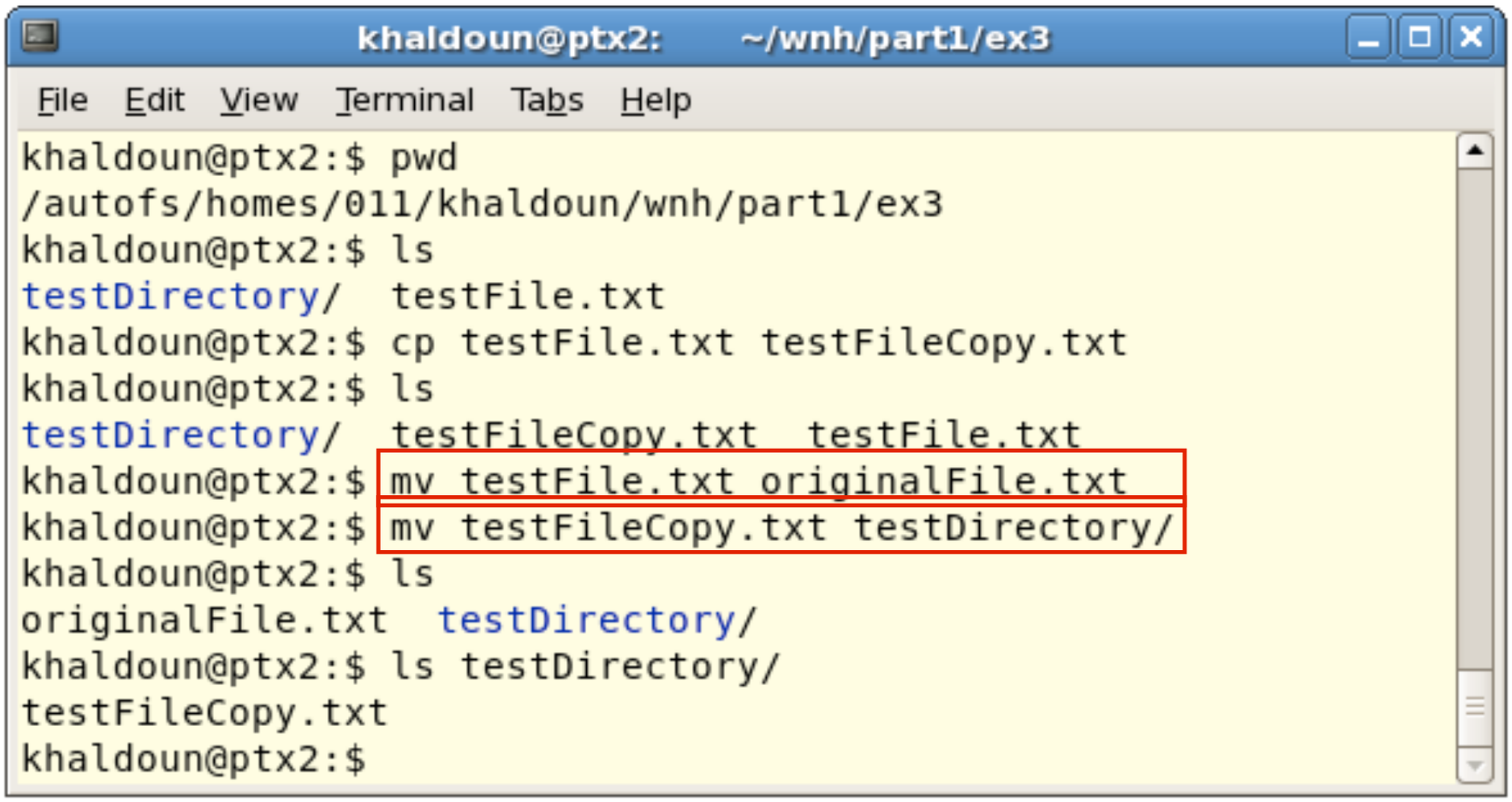
Example 3a: Directory commands

- **`pwd`** (print working directory)
- **`mkdir`** `dirname` (make directory, i.e. a folder)
- **`cd`** `dirname` (change directory, i.e. go to that folder)



Example 3b: Copying and moving

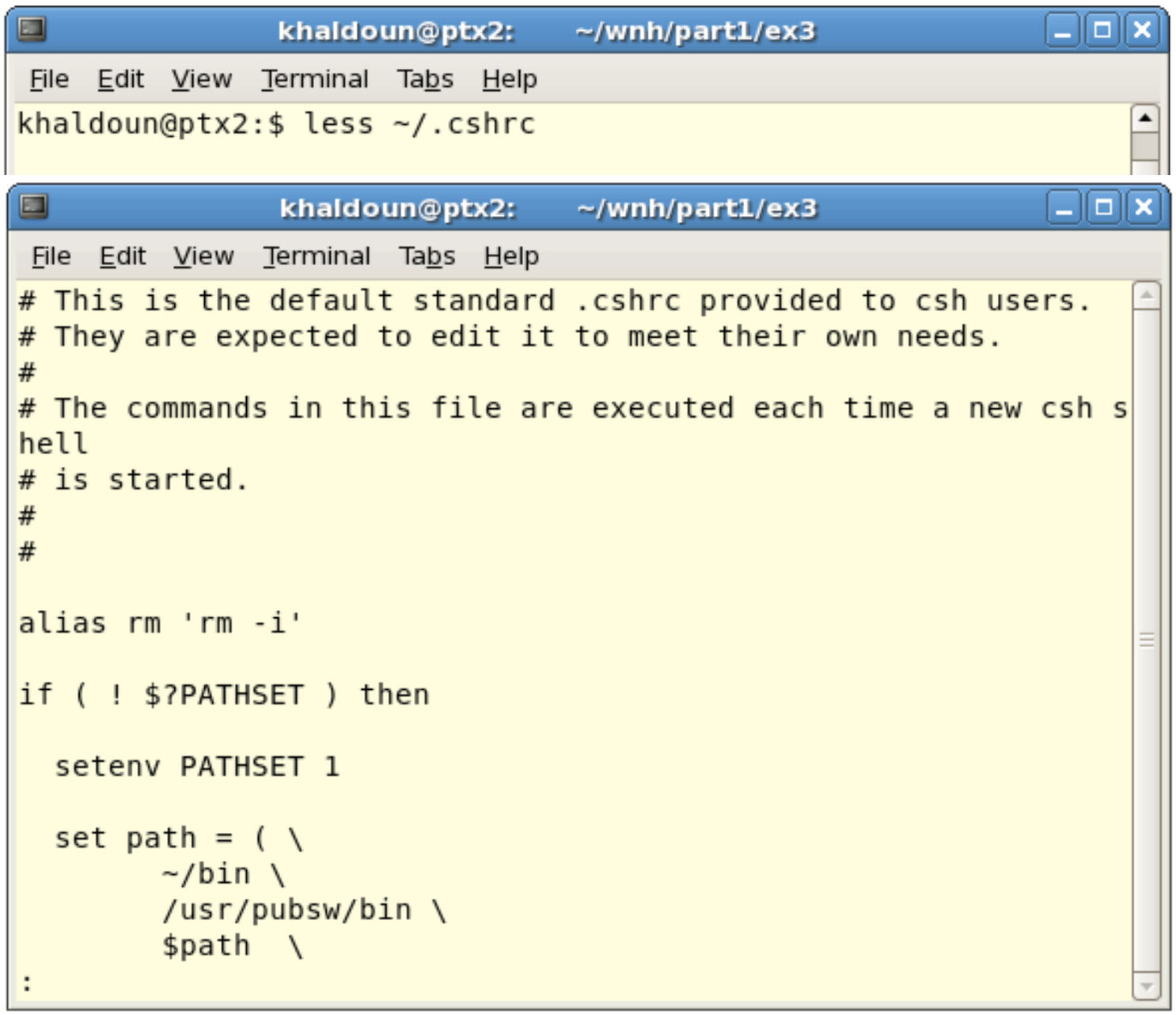
- **cp** source destination (copy files/dirs)
- **mv** source destination (move or rename)



Note the difference between the two move commands. The first one gives a new file name, which makes the move command into a rename command. The second one gives a directory as the last argument, which makes the move command move the actual file into that directory.

Example 3c: `less` and `more`

- **`less`/`more`** `textfilename` **(display contents of file)**



`more` allows you to view text files (no editing). It dumps out the contents of the file to the terminal window.

`less` performs the same function, but it more powerful and better suited for longer files (scrolling, searching, etc).

Getting around a file using `less` works like it does when using `man` (actually it's more correct to say that `man` uses `less` to display content, which is why you can use the same commands to get around):

Note: to scroll by a full page in `man`, hit SPACE; to scroll by one line, use the up and down arrow keys. To scroll backwards by a full page, hit the letter `b`. To quit and return to the command line, type `q`. To search for a phrase, type `/`, then type the query, then type ENTER. While in search mode, hit `n` to go to the next match, and `p` to go to the previous match.

Example 3d: Useful info

`.`	Current directory
`..`	Parent directory (up one level)
`~`	User's home directory

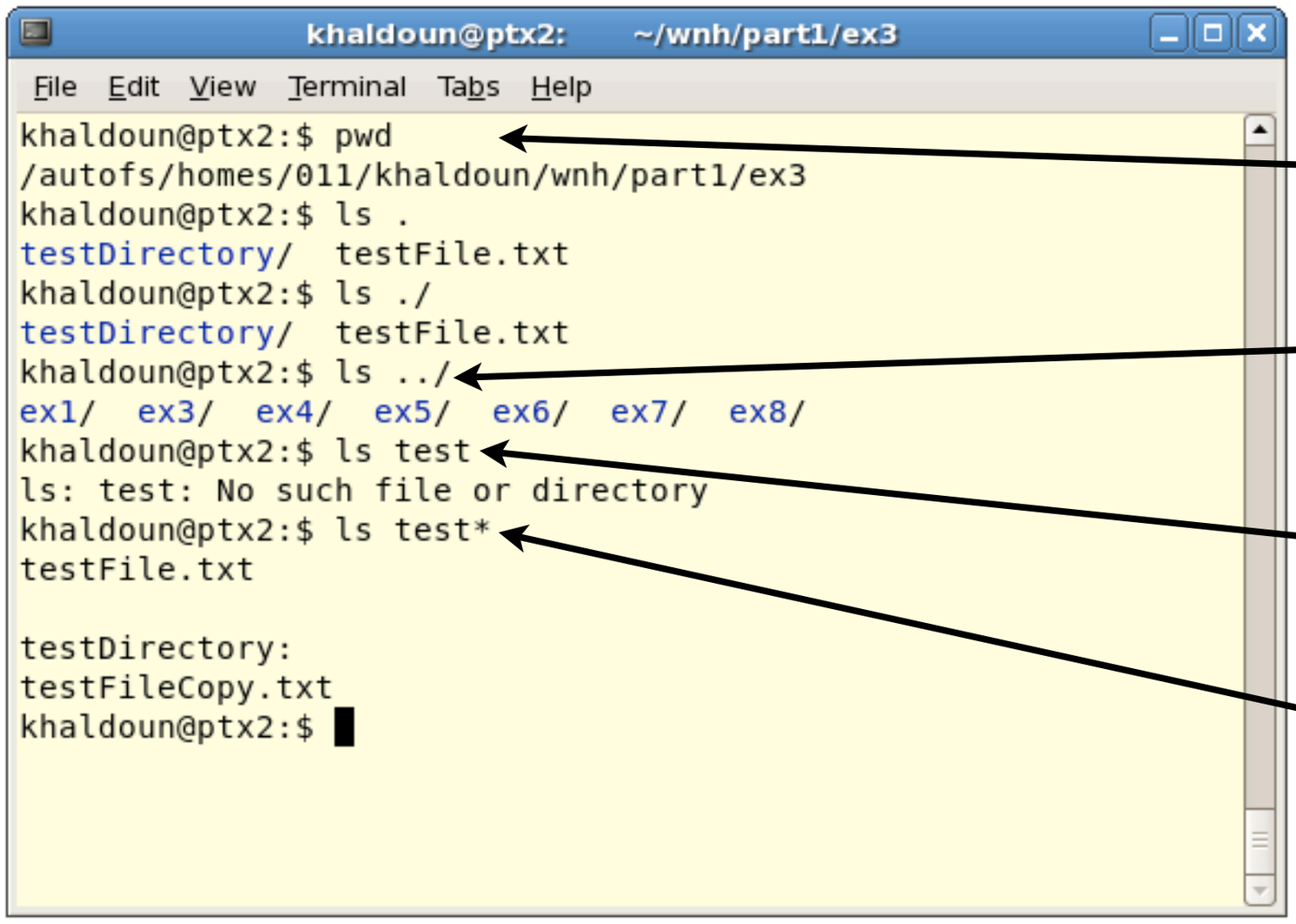
often used as `./` `../` and `~/` since the forward slash denotes separation between directories in Unix paths

`*`	matches any number of any characters
`?`	matches one of any character
`[abc]`	matches `a` or `b` or `c`

Also useful: the TAB key autocompletes

These are just a few of the useful things to know about typing commands in the command line.

Example 3f: Testing out useful info



Current directory (ex3)

Parent directory

No file called "test"

One file and one directory that match test* (meaning "test" followed by anything, including followed by nothing)

On TAB autocompletion: it is not necessary to type "`ls testFile`" to list this file. It's enough to type "`ls testF`" and then press the TAB key. The command line autocompletes to the only completion available. If you type "`ls t`" then TAB, it will autocomplete to "`test`" and then wait for user input to differentiate between testFile and testDirectory.

The parent directory is the one that contains the current directory. So if you create a directory called whynhow (as I did) and then create inside it directories called ex1, ex2, etc, then whynhow is the parent directory or ex1, ex2, etc.

Example 4: `grep` and `find`

`grep` and `find` are good examples of the power of the tools you'll typically use on Linux. They both become very powerful as you learn to use their options, but start out as relatively straightforward pattern-matching tools.

- grep searches for string (i.e. text) matches inside files

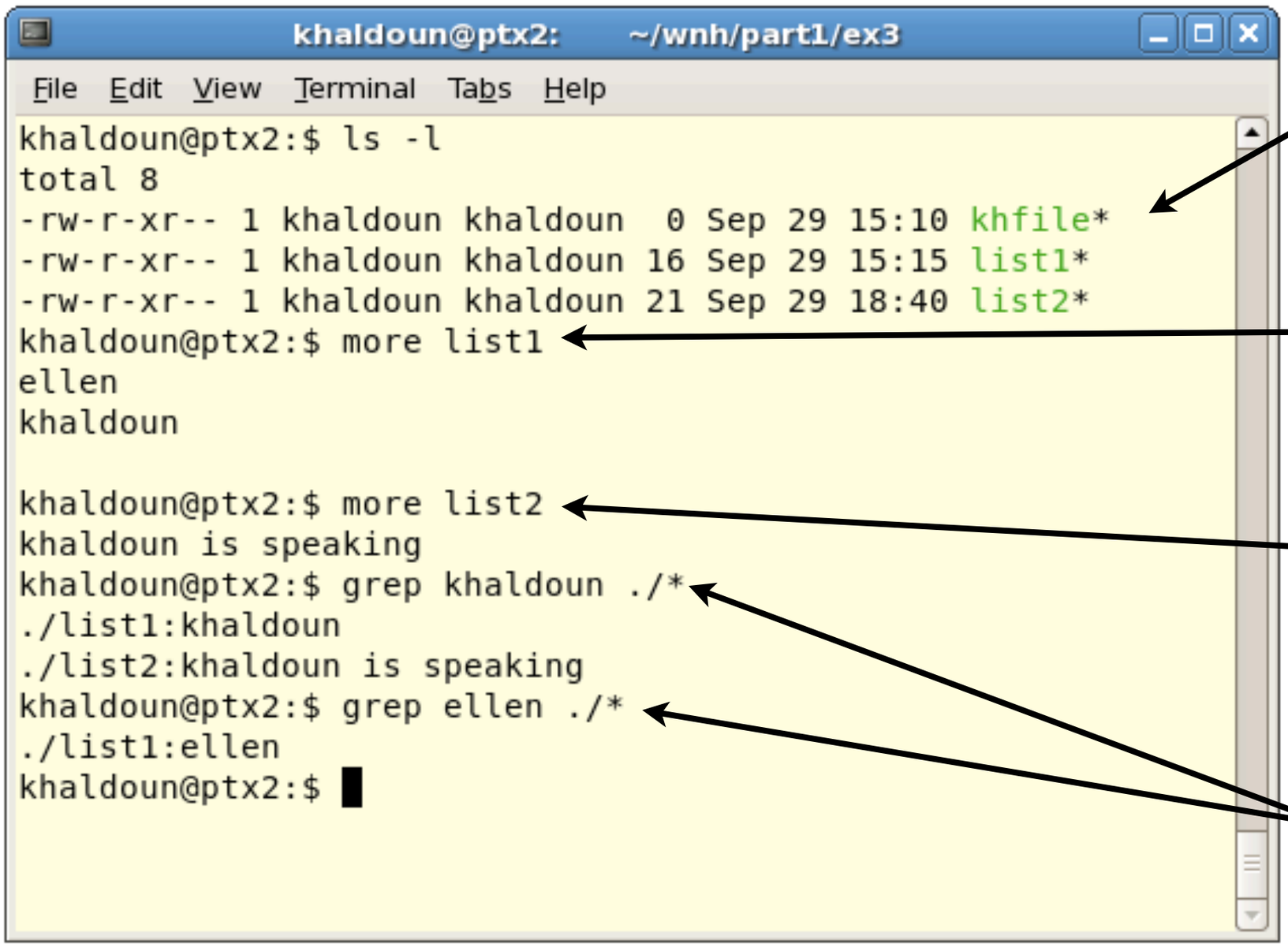
  ```
  grep pattern filelist
  ```

- find searches for files matching certain conditions:

  ```
  find directory -name 'filename'
  ```


Note the difference between `grep` and `find` in syntax. `grep` places the files to be searched after the pattern, whereas `find` first specifies the directory. Besides the order, note that `grep` asks for a file list (so all the files in the present directory would be `./*`), whereas `find` asks for just a directory (`./`).

Example 4: `grep` to find pattern in file



3 files in this directory

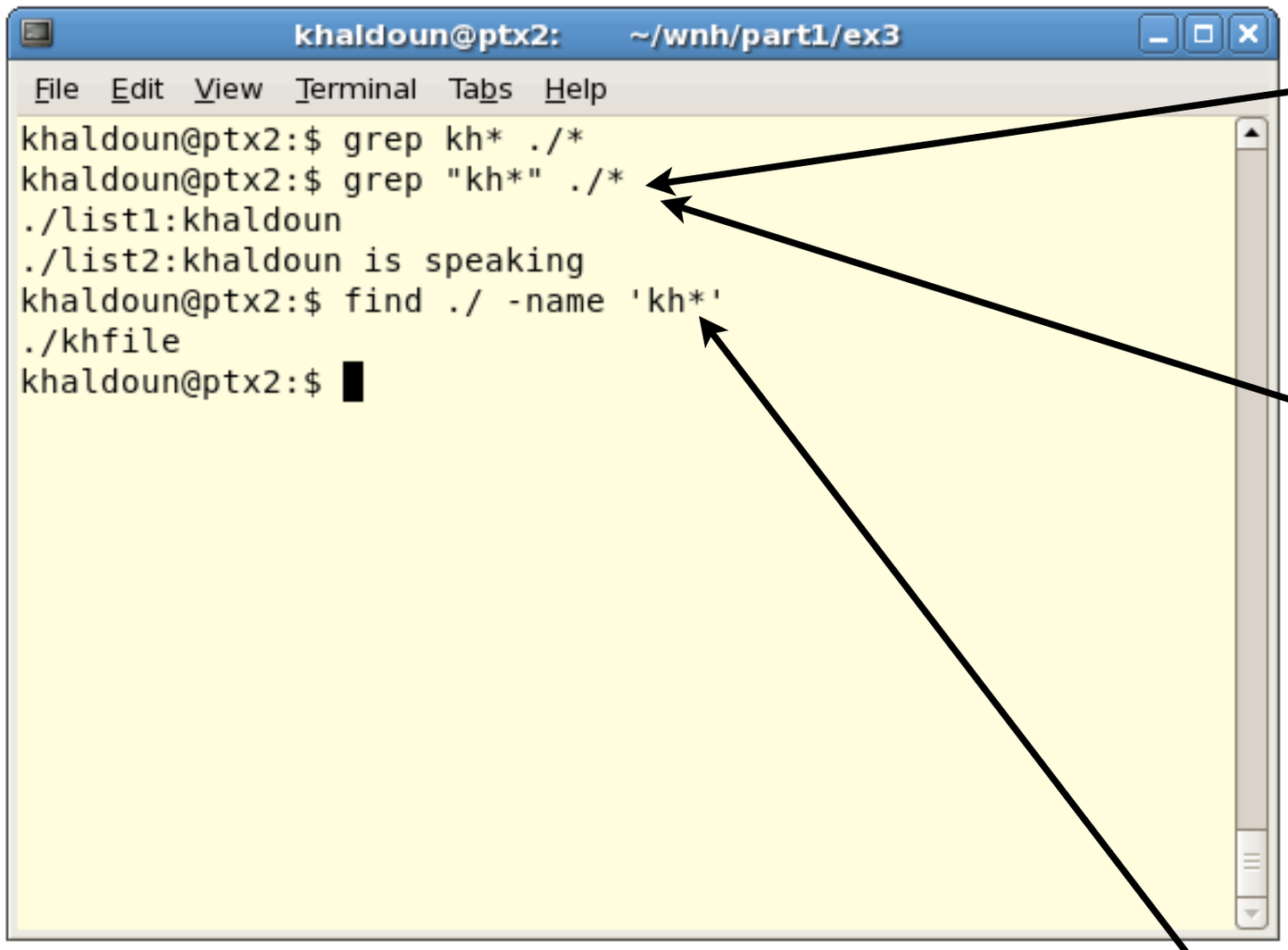
First file contains 2 names and 1 empty line (shown by more)

Second file contains 1 line

`grep` commands to search for these patterns in all the files that are in the current directory

Being able to match patterns from inside a file is extremely useful, especially once you include matching conditions using *, ?, and other matching syntax. You'll be going through log files and code considerably faster than you would otherwise.

Example 4: `find` to find files



Quotation marks are required for pattern matching or the search will fail

Note that grep searches the **contents** of files, and will not match a file that has the searched pattern only its filename (here khfile)

Use `find` to search for patterns in file names, or files times, or many many other file attributes (check the `man` page!)

In addition to these simple examples, you can tailor these tools to your liking. Use `grep` with the `-v` option to invoke anti-matching: it will find the lines that do not match the specified pattern. Use `find` with time specifiers to find files older than n minutes or newer than m days. And much much more. Check the `man` pages!

Intro to shell scripting

Scripting basics

- A script is a sequence of commands stored in a text file that can be run like any other command
- The use of programming constructs such as variables, loops and conditional statements make this more powerful than just a saved list of commands

At first, a script is useful because it saves you the trouble of typing in the commands you need repeatedly. If you find yourself performing the same series of steps over and over (say on several data sets), it's not only more convenient, but also better for the reproducibility of your experiment & analysis to write this series of steps into a script, and then simply run the script.

But the true power of scripting lies in the fact that it enables the use of important algorithmic & programming contructs (with little user overhead such as compilation of code, etc). If your work requires loops and conditional statements using command line commands, scripting isn't simply a convenience; it's the only way to get your work done.

Example 5: a backup script

Type this into a file called backup.csh

```
#!/bin/csh

# comment here: very basic backup

cd parentdirectory

rsync -avr originDir backupDir/
```

Then make it executable & run it!

```
chmod u+x ./backup.csh

./backup.csh
```


This will demonstrate the simple command list version of a script.

One of the most important computing habits to develop is the use of regular backups. So we'll demonstrate putting together a very simple backup script. This script will copy some data from a directory called originDir (modify for your own needs) to a destination called backupDir. This very simple backup overwrites any previous backup in the destination directory. In other words, any files which have changed in the origin will replace the older files in the destination. However, it will not delete files from the destination if they have been deleted from the source. The options used for `rsync` are: `-a` for archive mode (preserve time stamps, file attribures, etc), `-v` for verbose so that we see output on the terminal screen of what `rsync` is doing at all times, and `-r` to recursively enter directories and sync everything inside them as well.

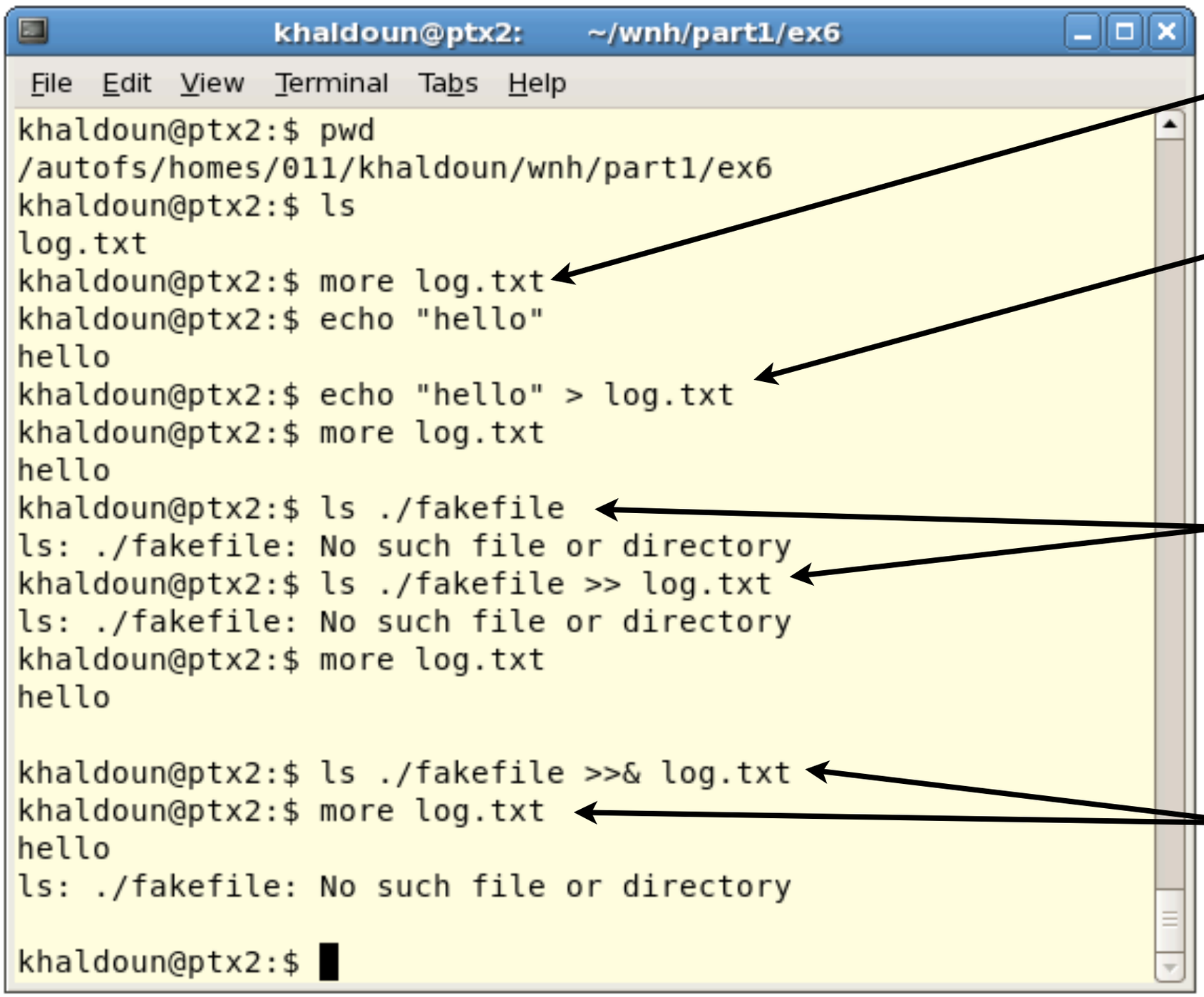
After we have written a file called `backup.csh`, we have to specify that this file is now executable (i.e. not just readable – for viewing, and writeable – for modifying, but also executable like any other command). We do so with the `chmod` command. The syntax is: `u` for user permission (as opposed to group or other), `x` for executable, and `+` for add this permission (as opposed to remove it).

We then run the script using `./backup.csh`. We specify the location of the executable as "this directory" (using ./) or the system may not know where to find this now-brand-new command called "backup.csh".

Example 6: output redirection & pipes

- The output from command and any errors normally get dumped to the terminal screen
- It's useful to save them when running scripts so that you can examine if anything went wrong
- **`command`** `> somelogfile` will save the output of **`command`** into the file `somelogfile`
- **`command`** `>& somelogfile` will save the output of AND any errors resulting from **`command`** into the file `somelogfile`
- `>>` and `>>&` append to the file `somelogfile` instead of replacing it
- You can also pipe the output of one command to be the input of another command using `|` (SHIFT-backslash on most keyboards). See example using `tee` and `wc`

Example 6: output redirection & pipes



`log.txt` is empty to start

We redirect the output of the `echo` command into `log.txt`, and check the content

We try a command we know will give an error, but >> does not seem to redirect to the log file

The use of >>& allows us to redirect for the normal output and the error output

Example 7: input parameters

- You can pass input parameters to your script just like you would to other commands:
  `myscript` `param1 param2`
- Inside the script, these parameters are referenced with `$1 $2` etc
- Although it's needless complication for the simple backup script, we'll use this for origin & destination to demonstrate

Example 7: input parameters

Type this into a file called backup_prep.csh

```
#!/bin/csh
set origin = $1
set destination = $2
echo ""
echo "the directory $origin will be backed up to $destination"
```

Example 8: Looping

- Two ways to loop: `foreach` and `while`
- `foreach` is demonstrated here

```
#!/bin/csh

foreach flipangle (30 60 90 120)

   set cmd = (ls -l data_flip${flipangle})

   echo $cmd

   eval $cmd

end
```

Credit to A. Stevens for exposure to the very useful "eval"

Example 8: Looping

khaldoun@gate: /autofs/space/ptx2_001/users/w...

```
khaldoun@gate:$
khaldoun@gate:$ pwd
/autofs/space/ptx2_001/users/whynhow/ex7
khaldoun@gate:$ ./loop_script.csh
the command evaluated will be:  ls -l data_flip30
-rw-rw-r--  1 khaldoun ptx 0 Dec  2 15:52 data_flip30
the command evaluated will be:  ls -l data_flip60
-rw-rw-r--  1 khaldoun ptx 0 Dec  2 15:52 data_flip60
the command evaluated will be:  ls -l data_flip90
-rw-rw-r--  1 khaldoun ptx 0 Dec  2 15:52 data_flip90
the command evaluated will be:  ls -l data_flip120
ls: data_flip120: No such file or directory
khaldoun@gate:$ ls -l
total 4
-rw-rw-r--  1 khaldoun ptx   0 Dec  2 15:52 data_flip30
-rw-rw-r--  1 khaldoun ptx   0 Dec  2 15:52 data_flip60
-rw-rw-r--  1 khaldoun ptx   0 Dec  2 15:52 data_flip90
-rwxrw-r--  1 khaldoun ptx 140 Dec  2 16:06 loop_script.csh
khaldoun@gate:$
```

The script loops through all the values listed in `foreach`, and executes a command each time.

The last value produces an error, since there is no file with the name data_flip120: an excellent time to have a log so that you can track how your script ran.

Conditional statements

- Structure of if statements is simple:

```
if (expression) then
    commands
    ...
else if (expression) then
    commands
    ...
    ...
else
    commands
    ...
endif
```

- The fun is in what you can put in `(expression)`

Conditional statements

- General logic and comparisons in expressions:
  - ! logical negation
  - && logical AND
  - || logical OR
  - == equals
  - != not equals
  - > < <= >= their usual math meanings

Conditional statements

- File operators
  - `-e file` true if file exists
  - `-d dir` true if dir exists and is a directory
  - `-z file` true if file exists and is zero size

- More at www.csem.duke.edu/Cluster/csh_basics.htm or at `man csh` ("File inquiry operators")

General Hints

- Always look at the manual page for any command you're not familiar with, or at the very least Google the command for some basic info.
- Searching man pages (and `less` output) is done with `/` followed by the search phrase followed by RETURN/ENTER. Cycling through results is done with `n` (next) and `p` (previous). Quitting is done with `q`.
- Keep track of learned commands and hints in a text file as you go along. Learning Linux/C shell/scripting really means learning, then forgetting, then relearning, etc.
- Don't hesitate to email if there are any questions arising from this discussion later on: `khaldoun@nmr.mgh.harvard.edu`