

Lecture 7: Shell Part VII

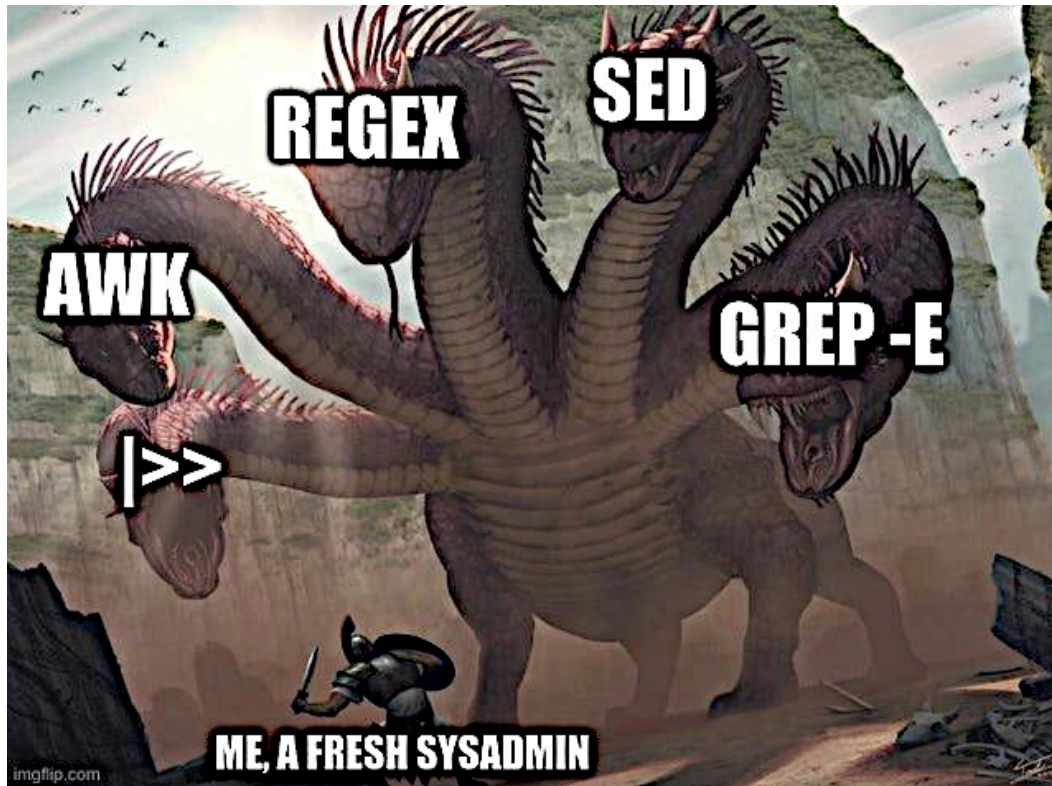
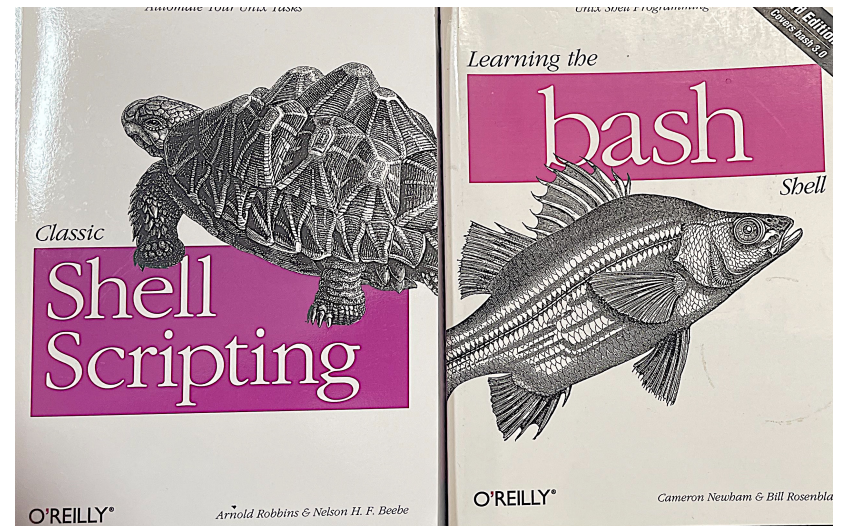


Photo of a new
Linux user learning
the command line



Suggested Literature

Both books are available at
<https://learning.oreilly.com>



RECAP: Sorting integers with **sort**

```
whovian@phy504:~/host$ more singers2.txt
4 Avril_Lavigne 100
10 Rihanna 50
2 Lady_Gaga 300
5 Usher 10
14 Shakira 1000
1 Justin_Timberlake 25
20 Billie_Joe_Armstrong 250
3 Taylor_Swift 500
whovian@phy504:~/host$ sort --key 3 -n singers2.txt
5 Usher 10
1 Justin_Timberlake 25
10 Rihanna 50
4 Avril_Lavigne 100
20 Billie_Joe_Armstrong 250
2 Lady_Gaga 300
3 Taylor_Swift 500
14 Shakira 1000
whovian@phy504:~/host$ sort --key 3 -nr singers2.txt
14 Shakira 1000
3 Taylor_Swift 500
2 Lady_Gaga 300
20 Billie_Joe_Armstrong 250
4 Avril_Lavigne 100
10 Rihanna 50
1 Justin_Timberlake 25
5 Usher 10
```

Added a third column
with the amount of
money you spent on
singer's albums

RECAP: Sorting dictionary ordering

Sorting
using
multiple
columns

```
whovian@phy504:~/host$ sort --key 1 sorting2.txt
```

```
a 1 1.0
a 13 2.1
a 4 0.001
a 6 0.01
a 8 0.35
b 12 3.1
b 2 0.1
b 7 0.01
b 9 2.3
c 10 0.1
c 11 1.0
c 3 0.3
c 5 0.5
```

```
whovian@phy504:~/host$ sort --key 1,1d --key 3,3g sorting2.txt
```

```
a 4 0.001
a 6 0.01
a 8 0.35
a 1 1.0
a 13 2.1
b 7 0.01
b 2 0.1
b 9 2.3
b 12 3.1
c 10 0.1
c 3 0.3
c 5 0.5
c 11 1.0
```

- The argument `-k1,1d` means “create a key starting at column 1 and ending at column 1 in dictionary order”
- For the last key you could have written `--key 3`

RECAP: Removing duplicate words

Piping is extremely powerful: lets pip **sort** and **uniq** command
uniq command filters adjacent lines from input

- **-d** removes duplicates
- **-u** only print unique lines
- **-c** count the records

```
whovian@phy504:~/host$ sort portugues-numbers | uniq -d
dez
oito
quatro
```

```
whovian@phy504:~/host$ sort portugues-numbers | uniq -c
1 cinco
2 dez
1 dois
1 nove
3 oito
2 quatro
1 seis
1 sete
1 tres
1 um
```

RECAP: Sorting with floating numbers

Did you notice the **-g** option on last slide?

- When dealing with float point **-g** is preferable
- When using scientific notation **-g** is mandatory
- [Awesome stackoverflow answer with in-depth explanation](#)

```
whovian@phy504:~/host$ printf '%s\n' 0.1 10 1e-2 | sort -n
0.1
1e-2
10
whovian@phy504:~/host$ printf '%s\n' 0.1 10 1e-2 | sort -g
1e-2
0.1
10
```

Why not using **-g** all the time instead of **-n**? Speed!

RECAP: counting words on Hamlet

An illustrative example of the power of shell commands

```
GNU nano 4.8
#!/bin/bash

tr -cs A-Za-z '\n' |
tr A-Z a-z |
sort |
uniq -c |
sort -k1,1nr -k2 |
sed ${1:-25}q
```

- Replace nonletters with newlines
- Map uppercase to lowercase
- Sort the words in ascending order
- Eliminate duplicates, showing their counts
- Sort by descending count, then by ascending word
- only line we don't understand (next class): we do understand `${1:-25}`

sed Xq: quit processing after X lines (optional command)

RECAP: counting words on Hamlet

How many words are in Hamlet?

```
whovian@phy504:~/host$ bash wf.sh 999999999999999 < hamlet.txt | wc -l  
4562
```

Least-frequent words on Hamlet?

```
whovian@phy504:~/host$ bash wf.sh 99999999 < hamlet.txt | tail -n 12 | pr --columns=4 -t -w80  
1 yaw                1 yeoman             1 yielding           1 younger  
1 yawn               1 yesterday          1 yon                1 yourselves  
1 yeasty             1 yesternight        1 yond               1 zone
```

Most-frequent words on Hamlet?

```
whovian@phy504:~/host$ bash wf.sh 12 < hamlet.txt | pr --columns=4 -t -w80  
1118 the             677 of               559 you              466 in  
987 and              635 i                 516 my                420 it  
747 to               566 a                  474 hamlet            407 that
```

RECAP: **sed** – string substitution

sed is a small power string editor program.

In combination with **regular expressions**,

sed has saved my research a coupled of times

Let's start simple. Consider this file

```
GNU nano 4.8
John Daggett, 341 King Road, Plymouth MA
Alice Ford, 22 East Broadway, Richmond VA
Orville Thomas, 11345 Oak Bridge Road, Tulsa OK
Terry Kalkas, 402 Lans Road, Beaver Falls PA
Eric Adams, 20 Post Road, Sudbury MA
Hubert Sims, 328A Brook Road, Roanoke VA
Amy Wilde, 334 Bayshore Pkwy, Mountain View CA
Sal Carpenter, 73 6th Street, Boston MA
```

Nice documentation: <https://www.gnu.org/software/sed/manual/>

RECAP: **sed** – string substitution

↓ What about multiple substitutions? Use option **-e**

```
wovian@phy504:~/host$ sed -e 's/ MA/, Massachusetts/'  
-e 's/ PA/, Pennsylvania/' address_list.txt  
John Daggett, 341 King Road, Plymouth, Massachusetts  
Alice Ford, 22 East Broadway, Richmond VA  
Orville Thomas, 11345 Oak Bridge Road, Tulsa OK  
Terry Kalkas, 402 Lans Road, Beaver Falls, Pennsylvania  
Eric Adams, 20 Post Road, Sudbury, Massachusetts  
Hubert Sims, 328A Brook Road, Roanoke VA  
Amy Wilde, 334 Bayshore Pkwy, Mountain View CA  
Sal Carpenter, 73 6th Street, Boston, Massachusetts
```

- Look at the space before MA. Be careful with characters between the first **/** and the second **/**
- Output redirection (save to file) **sed -f sedscr list > newlist**

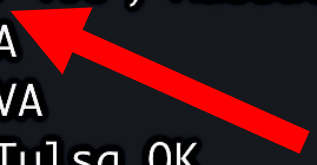
RECAP: **sed** – string substitution

sed can replace only lines that contain some key string

```
whovian@phy504:~/host$ sed -e '/Road/s/ MA/, Massachusetts/' address_list.txt
John Daggett, 341 King Road, Plymouth, Massachusetts
Alice Ford, 22 East Broadway, Richmond VA
Orville Thomas, 11345 Oak Bridge Road, Tulsa OK
Terry Kalkas, 402 Lans Road, Beaver Falls PA
Eric Adams, 20 Post Road, Sudbury, Massachusetts
Hubert Sims, 328A Brook Road, Roanoke VA
Amy Wilde, 334 Bayshore Pkwy, Mountain View CA
Sal Carpenter, 73 6th Street, Boston MA
```

sed only replaces **MA** with “**, Massachusetts**” on lines that contain **Road**

```
whovian@phy504:~/host$ sed -e '/Road/!s/ MA/, Massachusetts/' address_list.txt
John Daggett, 341 King Road, Plymouth MA
Alice Ford, 22 East Broadway, Richmond VA
Orville Thomas, 11345 Oak Bridge Road, Tulsa OK
Terry Kalkas, 402 Lans Road, Beaver Falls PA
Eric Adams, 20 Post Road, Sudbury MA
Hubert Sims, 328A Brook Road, Roanoke VA
Amy Wilde, 334 Bayshore Pkwy, Mountain View CA
Sal Carpenter, 73 6th Street, Boston, Massachusetts
```



! = opposite behavior. Replace only on lines that don't contain Road

sed - Suppressing display of input lines

Default operation of **sed** is to output every input line (except with **-n**). With **-n**, each instruction intended to produce an output must contain the print option, **“/p”**.

```
whovian@phy504:~/host$ sed -n -e 's/MA/Massachusetts/p'
address_list.txt
John Daggett, 341 King Road, Plymouth Massachusetts
Eric Adams, 20 Post Road, Sudbury Massachusetts
Sal Carpenter, 73 6th Street, Boston Massachusetts
```



Option	Description
--------	-------------

-e	Editing instruction follows.
-----------	------------------------------

-f	Filename of script follows.
-----------	-----------------------------

-n	Suppress automatic output of input lines.
-----------	---

Without p: no output at all
with p, output only modified
lines

sed - Suppressing display of input lines

With **-n**, each instruction intended to produce output must contain a print command, **/p**. We can use that to only print lines with (or w/o with the **!** option) a special word

```
whovian@phy504:~/host$ sed -n '/MA/p' address_list.txt
John Daggett, 341 King Road, Plymouth MA
Eric Adams, 20 Post Road, Sudbury MA
Sal Carpenter, 73 6th Street, Boston MA
```



```
whovian@phy504:~/host$ sed -n '/MA/!p' address_list.txt
Alice Ford, 22 East Broadway, Richmond VA
Orville Thomas, 11345 Oak Bridge Road, Tulsa OK
Terry Kalkas, 402 Lans Road, Beaver Falls PA
Hubert Sims, 328A Brook Road, Roanoke VA
Amy Wilde, 334 Bayshore Pkwy, Mountain View CA
```

sed with 'i' - writing lines

sed can be used to insert new text in a file

```
whovian@phy504:~/host$ sed -e '2i Vivian Miranda, Jamaica, New York City, NY' address_list.txt
John Daggett, 341 King Road, Plymouth MA
Vivian Miranda, Jamaica, New York City, NY
Alice Ford, 22 East Broadway, Richmond VA
Orville Thomas, 11345 Oak Bridge Road, Tulsa OK
Terry Kalkas, 402 Lans Road, Beaver Falls PA
Eric Adams, 20 Post Road, Sudbury MA
Hubert Sims, 328A Brook Road, Roanoke VA
Amy Wilde, 334 Bayshore Pkwy, Mountain View CA
Sal Carpenter, 73 6th Street, Boston MA
```



sed can also be used to just replace a line

```
whovian@phy504:~/host$ sed -e '3c Vivian Miranda, Jamaica, New York City, NY' address_list.txt
John Daggett, 341 King Road, Plymouth MA
Alice Ford, 22 East Broadway, Richmond VA
Vivian Miranda, Jamaica, New York City, NY
Terry Kalkas, 402 Lans Road, Beaver Falls PA
Eric Adams, 20 Post Road, Sudbury MA
Hubert Sims, 328A Brook Road, Roanoke VA
Amy Wilde, 334 Bayshore Pkwy, Mountain View CA
Sal Carpenter, 73 6th Street, Boston MA
```



sed is so versatile. More functionality

sed can be used to delete lines that contain a key word

```
whovian@phy504:~/host$ sed '/MA/d' address_list.txt
Alice Ford, 22 East Broadway, Richmond VA
Orville Thomas, 11345 Oak Bridge Road, Tulsa OK
Terry Kalkas, 402 Lans Road, Beaver Falls PA
Hubert Sims, 328A Brook Road, Roanoke VA
Amy Wilde, 334 Bayshore Pkwy, Mountain View CA
```

Print the line numbers of lines containing a key word

```
whovian@phy504:~/host$ sed -n '/MA/=' address_list.txt
1
5
8
```

sed is so versatile. More functionality

sed commands summary

The following commands are supported in GNU sed. Some are standard POSIX commands, while other are GNU extensions. Details and examples for each command are in the following sections. (Mnemonics) are shown in parentheses.

a

text

Append *text* after a line.

a text

Append *text* after a line (alternative syntax).

b label

Branch unconditionally to *label*. The *label* may be omitted, in which case the next cycle is started.

c

text

Replace (change) lines with *text*.

c text

Replace (change) lines with *text* (alternative syntax).

d

Delete the pattern space; immediately start next cycle.

and much more...

<https://www.gnu.org/software/sed/manual/sed.html#sed-commands-list>

sed global option

caveat substitution commands – the global option **g**

```
whovian@phy504:~/host$ sed -e '2i Vivian Miranda, Jamaica, New York City NY NY'
address_list.txt | sed -e 's/ NY/, New York/'
John Daggett, 341 King Road, Plymouth MA
Vivian Miranda, Jamaica, New York City, New York NY
Alice Ford, 22 East Broadway, Richmond VA
Orville Thomas, 11345 Oak Bridge Road, Tulsa OK
Terry Kalkas, 402 Lans Road, Beaver Falls PA
Eric Adams, 20 Post Road, Sudbury MA
Hubert Sims, 328A Brook Road, Roanoke VA
Amy Wilde, 334 Bayshore Pkwy, Mountain View CA
Sal Carpenter, 73 6th Street, Boston MA
```

without **-g**, sed only
do one substitution
per line

```
whovian@phy504:~/host$ sed -e '2i Vivian Miranda, Jamaica, New York City NY NY'
address_list.txt | sed -e 's/ NY/, New York/g'
John Daggett, 341 King Road, Plymouth MA
Vivian Miranda, Jamaica, New York City, New York, New York
Alice Ford, 22 East Broadway, Richmond VA
Orville Thomas, 11345 Oak Bridge Road, Tulsa OK
Terry Kalkas, 402 Lans Road, Beaver Falls PA
Eric Adams, 20 Post Road, Sudbury MA
Hubert Sims, 328A Brook Road, Roanoke VA
Amy Wilde, 334 Bayshore Pkwy, Mountain View CA
Sal Carpenter, 73 6th Street, Boston MA
```

sed global option

Replacing tabs with spaces

First example replaced tabs with letter A
(just to check)

```
whovian@phy504:~/host/Phy504Docker$ echo -e "1\t2\t3" | sed -e 's/\t/A/g'
1A2A3
whovian@phy504:~/host/Phy504Docker$ echo -e "1\t2\t3" | sed -e 's/\t/ /g'
1 2 3
```

Regular Expressions

Power and useful features to deal with strings

Regex: so useful when combined with **egrep** and **sed**

```
whovian@phy504:~/host$ egrep --help
Usage: grep [OPTION]... PATTERNS [FILE]...
Search for PATTERNS in each FILE.
Example: grep -i 'hello world' menu.h main.c
PATTERNS can contain multiple patterns separated by newlines.
```

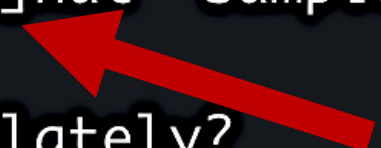
```
Hey Phy504
What are you doing?
Really, what have you been doing lately?
Heyyyyyyy, what is the nature of Physics?
What is the microphysics of dark energy?
Xhat is the UV physics of the Universe?
1hat is the IR physics of the Universe?
123hat is the EFT physics of the Universe?
```

Sample text

Regular Expressions

Regex: so useful when combined with **egrep** and **sed**
[...] is a metacharacter in regex

```
whovian@phy504:~/host$ egrep '[Ww]hat' sample_text_egrep.txt
What are you doing?
Really, what have you been doing lately?
Heyyyyyyy, what is the nature of Physics?
What is the microphysics of dark energy?
```



.

Matches any *single* character except newline

*

Matches any number (including zero) of the single character (including a character specified by a regular expression) that immediately precedes it.

[...]

Matches any *one* of the class of characters enclosed between the brackets. A circumflex (^) as first character in-

Regular Expressions

\$	As last character of regular expression, matches the end of the line. Matches the end of a string in awk, even if the string contains embedded newlines.
+	Matches one or more occurrences of the preceding regular expression.
?	Matches zero or one occurrences of the preceding regular expression.
	Specifies that either the preceding or following regular expression can be matched (alternation).
()	Groups regular expressions.
\{n,m\}	Matches a range of occurrences of the single character (including a character specified by a regular expression) that immediately precedes it. \{n\} will match exactly <i>n</i> occurrences, \{n,\} will match at least <i>n</i> occurrences, and \{n,m\} will match any number of occurrences between <i>n</i> and <i>m</i> . (sed and grep only, may not be in some very old versions.)
\	Escapes the special character that follows.

Regular Expressions

Metacharacters in Regex

```
whovian@phy504:~/host$ egrep '.hat' sample_text_egrep.txt
```

What are you doing?

Really, what have you been doing lately?

Heyyyyyyy, what is the nature of Physics?

What is the microphysics of dark energy?

Xhat is the UV physics of the Universe?

1hat is the IR physics of the Universe?

123hat is the EFT physics of the Universe?

```
whovian@phy504:~/host$ egrep '^ [1-9]hat' sample_text_egrep.txt
```

1hat is the IR physics of the Universe?

```
whovian@phy504:~/host$ egrep '[1-9]hat' sample_text_egrep.txt
```

1hat is the IR physics of the Universe?

123hat is the EFT physics of the Universe?

```
whovian@phy504:~/host$ egrep '[A-Z]hat' sample_text_egrep.txt
```

What are you doing?

What is the microphysics of dark energy?

Xhat is the UV physics of the Universe?

```
whovian@phy504:~/host$ egrep '[a-z]hat' sample_text_egrep.txt
```

Really, what have you been doing lately?

Heyyyyyyy, what is the nature of Physics?

Regular Expressions

Question: Why did both expressions match **123hat**?

```
whovian@phy504:~/host$ egrep "[1-9]+hat" sample_text_egrep.txt
1hat is the IR physics of the Universe?
123hat is the EFT physics of the Universe?
whovian@phy504:~/host$ egrep "[1-9]hat" sample_text_egrep.txt
1hat is the IR physics of the Universe?
123hat is the EFT physics of the Universe?
```

The regex **[1-9]hat** matched **3hat**. Let's use the “**^**” to prove that

```
whovian@phy504:~/host$ egrep "^[1-9]hat" sample_text_egrep.txt
1hat is the IR physics of the Universe?
whovian@phy504:~/host$ egrep "^[1-9]+hat" sample_text_egrep.txt
1hat is the IR physics of the Universe?
123hat is the EFT physics of the Universe?
```

^ = match from the beginning of the string / line.

So, you need multiple digits (**+**) to go to hat!

Regular Expressions

Be careful. The position of symbols matters!

```
whovian@phy504:~/host$ egrep '[^1-9]hat' sample_text_egrep.txt
What are you doing?
Really, what have you been doing lately?
Heyyyyyyy, what is the nature of Physics?
What is the microphysics of dark energy?
Xhat is the UV physics of the Universe?
```



“[^]” here means all characters except **[1-9]**

Metacharacters: second example

GNU nano 4.8	
I can do it	whovian@phy504:~/host\$ egrep "can[no']t" sample_text_egrep2.txt
I cannot do it	I can't do it
I can not do it	whovian@phy504:~/host\$ egrep "can[no']*t" sample_text_egrep2.txt
I can't do it	I cannot do it
I cant do it	I can not do it
	I can't do it
	I cant do it



Regular Expressions

It is sometimes difficult to match a complete word.
If we want to match the pattern “**book**”, our search will hit the word “**book**” and “**books**” but also “**bookish,**” “**handbook,**” and “**booky.**”

```
book 1
books 2
bookish, 3
handbook, 4
booky. 5
book? 6
book, 7
book! 8
```

```
whovian@phy504:~/host$ egrep 'book' sample_text_egrep3.txt
```

```
book 1
books 2
bookish, 3
handbook, 4
booky. 5
```

```
whovian@phy504:~/host$ egrep ' book ' sample_text_egrep3.txt
```

```
whovian@phy504:~/host$ egrep 'book ' sample_text_egrep3.txt
book 1
```

**Both
wrong**



Regular Expressions

It is sometimes difficult to match a complete word. If we want to match the pattern “**book**”, our search will hit the word “book” and “**books**” but also “**bookish**,” “**handbook**,” and “**booky**.”

```
book 1
books 2
bookish, 3
handbook, 4
booky. 5
book? 6
book, 7
book! 8
```

```
whovian@phy504:~/host$ egrep 'books*' sample_text_egrep3.txt
book 1
books 2
whovian@phy504:~/host$ egrep 'book.*' sample_text_egrep3.txt
book 1
books 2
bookish, 3
handbook, 4
booky. 5
book? 6
book, 7
book! 8
whovian@phy504:~/host$ egrep 'book.?' sample_text_egrep3.txt
book 1
books 2
handbook, 4
book? 6
book, 7
book! 8
```



0 or 1 occurrences of the preceding char

Regular Expressions

It is sometimes difficult to match a complete word. If we want to match the pattern “**book**”, our search will hit the word “**book**” and “**books**” but also “**bookish**,” “**handbook**,” and “**booky**.”

```
book 1
books 2
bookish, 3
handbook, 4
booky. 5
book? 6
book, 7
book! 8
Xbook! 9
!{book}! 10
```

```
whovian@phy504:~/host$ egrep '\["[{(!)*book.?' sample_text_egrep3.txt
```

```
book 1
books 2
handbook, 4
book? 6
book, 7
book! 8
Xbook! 9
```

[] encapsulates "[{(!

Why does it match Xbook?

```
whovian@phy504:~/host$ egrep 'book[]})?!.,;:\s]*' sample_text_egrep3.txt
```

```
book 1
books 2
handbook, 4
book? 6
book, 7
book! 8
Xbook! 9
!{book}! 10
```

*** following the class means "match zero or more of these characters." Even if none of these characters are present, the regex will still match.**

Regular Expressions

It is sometimes difficult to match a complete word. If we want to match the pattern “**book**”, our search will hit the word “book” and “books” but also “bookish,” “handbook,” and “booky.”

```
whovian@phy504:~/host$ egrep '[^"][{}!]*book[]})?!.,;:\s]* ' sample_text_egrep3.txt
book 1
books 2
handbook, 4
book? 6
book, 7
book! 8
Xbook! 9
!{book}! 10
whovian@phy504:~/host$ egrep '[^"][{}!]*book[]})?!.,;:\s]* ' sample_text_egrep3.txt
whovian@phy504:~/host$
```

Second [] encapsulates
})?!.,;:\s

Almost!

```
whovian@phy504:~/host$ egrep ' (^| ) [^"][{}!]*book[]})?!.,;:\s]* ' sample_text_egrep3.txt
book 1
books 2
book? 6
book, 7
book! 8
!{book}! 10
```

^ or space

Finally!

Regular Expressions

Why (^|) was so important?

Why "Xbook" Does Not Match:

- When testing against "Xbook", consider where the match must begin:
 - Option 1:** At the beginning of the line (^).
 - Option 2:** After a space ().
- For "Xbook":
 - At the beginning:**
 - The match would start at the beginning of the string.
 - The pattern would try to match [\[{"{(!)*book.? starting at the first character.
 - However, the first character is "X", which is not in the character class [\[{"{(!).
 - Since that part can match zero characters, the regex then expects "book" to immediately appear.
 - But starting at the beginning, "Xbook" does not begin with "book" (it begins with "X").
 - After a space:**
 - There's no space in "Xbook" before "book".
 - The substring "book" does occur starting at position 1, but it's preceded by an "X", not a space.

```
whovian@phy504:~/host$ egrep '(^| )\[{"{(!)*book[?]*'
book 1
books 2
book? 6
book, 7
book! 8
!{book}! 10
```

Because of the leading group (^|), the regex requires that "book" either appears at the very start or right after a space. In "Xbook", "book" is preceded by "X", so neither condition is met.

regex + sed -E = power

In sed, we can (regex) to save the result of regex.

Then we can use \1, \2... to make replacements.

This is so powerful!

e.g, shuf -i 1-10 print permutation without replacement

```
whovian@phy504:~/host$ shuf -i 1-10
7
10
9
3
8
1
5
2
4
6
```

regex + sed -E = power

In sed -E, we can (regex) to save the result of regex.

Then we can use \1, \2... to make replacements.

e.g, shuf -i 1-10 print permutation without replacement

Let's add a minus sign to all output

```
whovian@phy504:~/host$ shuf -i 1-10 | sed -E 's/([0-9]+)/-\1/'
```

-3

-1

-6

-2

-10

-8

-7

-9

-5

-4

The result of the regex [0-9]+ is saved on variable \1. The parenthesis created a “capturing group”

regex + sed -E = power

```
whovian@phy504:~/host$ shuf -i 1-10 | egrep "^[0-9]$"
7
8
3
6
4
5
1
9
2
```

Challenge: what if we add to a minus sign only to single digits numbers?

```
whovian@phy504:~/host$ shuf -i 1-10 | sed -E 's/([0-9])/-\1/'
-5
-1
-6
-9
-8
-4
-2
-7
-3
10
```

We need to add the end of line regex `$` after `[0-9]` (one digit) .

The parenthesis created a “capturing group”

regex + sed -E = power

```
whovian@phy504:~/host/Phy504Docker$ sed -E 's/^(.{1,80}).*/\1/' address_list.txt
John Daggett, 341 King Road, Plymouth MA
Alice Ford, 22 East Broadway, Richmond VA
Orville Thomas, 11345 Oak Bridge Road, Tulsa OK
Terry Kalkas, 402 Lans Road, Beaver Falls PA
Eric Adams, 20 Post Road, Sudbury MA
Hubert Sims, 328A Brook Road, Roanoke VA
Amy Wilde, 334 Bayshore Pkwy, Mountain View CA
Sal Carpenter, 73 6th Street, Boston MA
```

```
whovian@phy504:~/host/Phy504Docker$ sed -E 's/^(.{1,10}).*/\1/' address_list.txt
John Dagge
Alice Ford
Orville Th
Terry Kalk
Eric Adams
Hubert Sim
Amy Wilde,
Sal Carpen
```



Line truncation

regex + sed -E = power

Remove all alphanumeric

```
whovian@phy504:~/host/Phy504Docker$ more test_sed_regex.txt
cat stdlist
#ID      #Name
[101]    -Ali
[102]    -Neha
whovian@phy504:~/host/Phy504Docker$ sed 's/[A-Za-z0-9]//g' test_sed_regex.txt

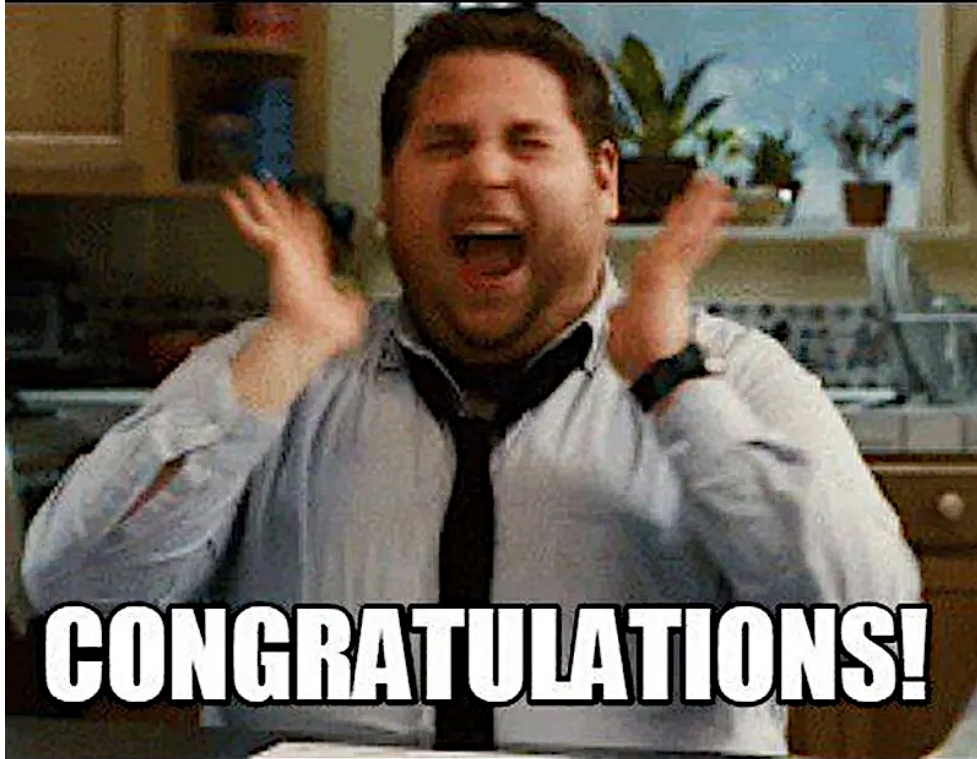
#      #
[]     -
[]     -
```

Replaces 4 consecutive space characters with tab character

```
whovian@phy504:~/host/Phy504Docker$ more test_sed_regex2.txt
1      2      3
4      5      6
7      9      10
whovian@phy504:~/host/Phy504Docker$ sed -e 's/    /\t/g' test_sed_regex2.txt
1      2      3
4      5      6
7      9      10
```

Congratulations

You now have a solid understanding of the Unix terminal, which can be helpful in research, such as when your pipeline needs to handle files.



Homework I

- Write a script that replaces all occurrences of the letter “a” with the letter “z” in a file
- Write a script that removes all whitespace characters (space, tab, newline) in a file
- Write a script that translates all digits to their corresponding words in a file (need to use SED)
- Write a script that displays the content of a file but replacing all spaces with newlines!
- Piping the pipeline from the exercise above, write a script that prints the number of unique words in a file. The display should start with the words that were repeated the most.
- Create a one-line pipe script that will sort the output of `echo "4 7 2 9 5"` numerically
- Create a one-line pipe script that will sort the output of `echo "John Smith, Jane Doe, Bob Johnson"` alphabetically by last name
- Create a one-line pipe script that will sort the following IP addresses by the last octet `echo "192.168.1.10. 192.168.1.3. 192.168.1.5"`

Homework II

- Write a script that replaces “old_word” with “new_word” in file.txt and output the result to new_file.txt.
- Write a script that deletes a line that contains Vivian word
- The command

```
history | sed -E 's/ *[0-9][0-9]* *//' | sort | uniq -c | sort -rn | head -10
```

list the most used bash commands by you. Explain how this program works

- Explain the difference of output between

```
history | sed -E 's/ *[0-9][0-9]* *//' | sort | uniq -c | sort -rn | head -10
```

and

```
history | sed -E 's/ [0-9][0-9]* *//' | sort | uniq -c | sort -rn | head -10
```

Homework III

- In sed, ‘**Xa new_line**’ is similar to ‘**Xi new_line**’ except that sed appends **new_line** after line X instead of inserting it at line X (check yourself!). So, write a script that appends the line **Vivian Miranda, Jamaica, New York City NY** after every occurrence of the word **Road** on address_list.txt file that I used on many examples
- Bash, any line that begins with **#** is a comment line. Write a sed script that deletes all comment lines. Hint: study regex **^**
- **shuf -i 1-10** prints the permutation without replacement of 1,...,10. Create a sed script that takes the output of **shuf -i 1-10** and change the sign of 10 only

Homework IV

- `shuf -i 49-80` prints the permutation without replacement of 49,...,80. Create a sed script that takes the output of `shuf -i 49-80` and change the sign of all numbers in the mathematical range [50,69]
- Explain the weird behavior of this program

```
whovian@phy504:~/host$ shuf -i 99-100 | sed -E 's/([0-9]{2}+)$/-\1/'
1-00
-99
whovian@phy504:~/host$ shuf -i 999-1000 | sed -E 's/([0-9]{2}+)$/-\1/'
9-99
-1000
```

- The following **regex** can catch e-mail addresses but has a problem/bug. Can you identify the problem and fix it?

```
whovian@phy504:~/host$ echo -e "vivian.miranda@stonybrook.edu" | egrep '[A-Za-z0-9._]+@[A-Za-z0-9.]+'
vivian.miranda@stonybrook.edu
```

Homework V

Test the following Regular Expressions with egrep

Item	Regular Expression
Postal Abbreviation for State	<code>[A-Z][A-Z]</code>  space
City, State	<code>^.*,[A-Z][A-Z]</code>
City, State, Zip (POSIX egrep)	<code>^.*,[A-Z][A-Z][0-9]{5}(-[0-9]{4})?</code>
Month, Day, Year	<code>[A-Z][a-z]\{3,9\}[0-9]\{1,2\},[0-9]\{4\}</code>
U.S. Social Security Number	<code>[0-9]\{3\}-[0-9]\{2\}-[0-9]\{4\}</code>
North-American Local Telephone	<code>[0-9]\{3\}-[0-9]\{4\}</code>
Formatted Dollar Amounts	<code>\\$[0-9]*\.[0-9][0-9]</code>

HW VI. The Bandit: a wargame levels 2/3

- [The Bandit](#) wargame is aimed at absolute beginners. It will teach the basics needed to be able to play other wargames.

Although OverTheWire is a reputable site, you must only run that HW inside the docker *phy504_2025_secure* container. The VM will protect you by isolating itself from the host machine

- **Level 2:** The password for the next level is stored in a file called **-** located in the home directory.

Commands you may need to solve level 2: **ls, cd, cat, file, du, find**

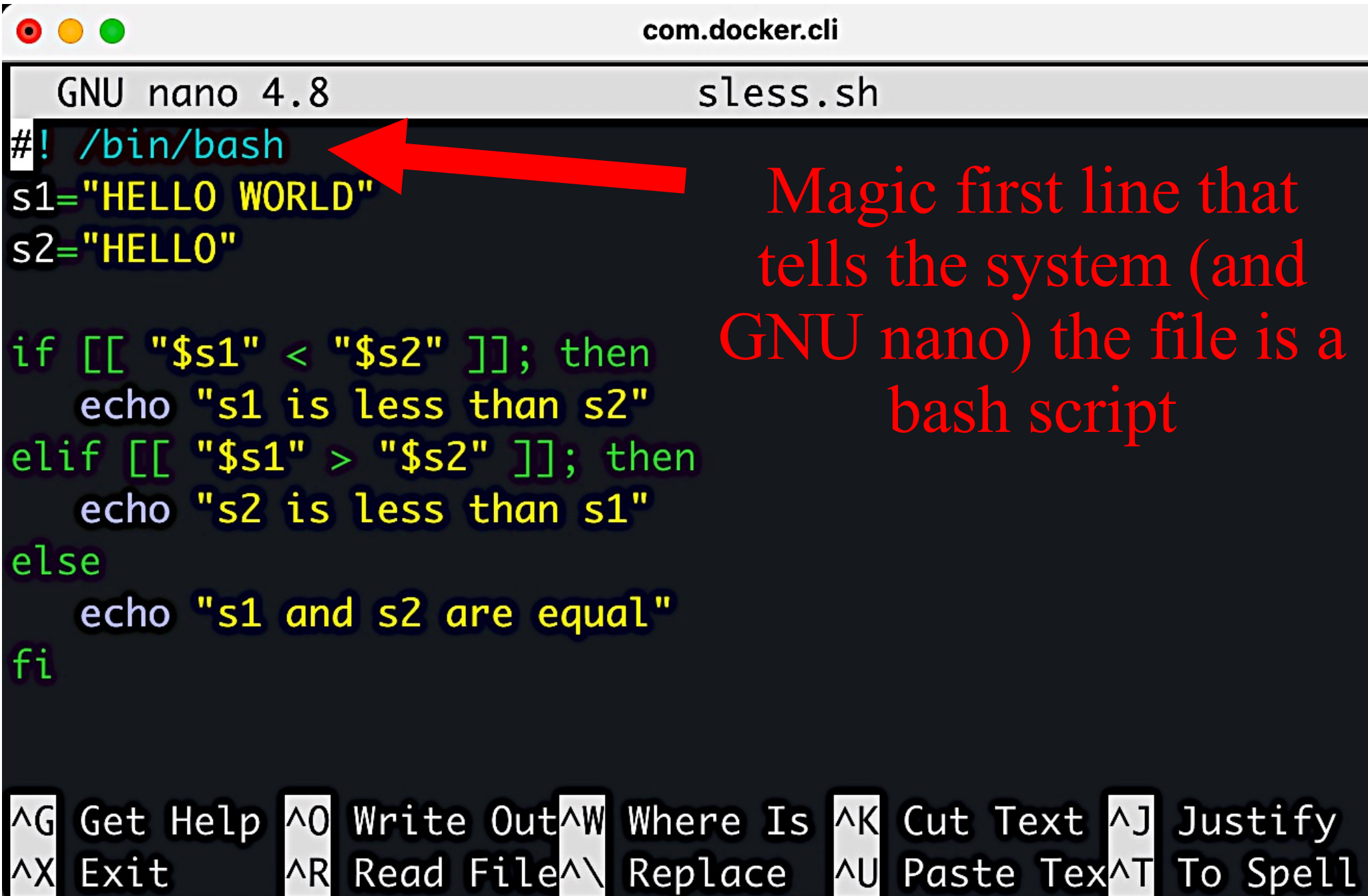
- **Level 3:** The password for the next level is stored in a file called **spaces in this filename** located in the home directory

Commands you may need to solve level 3: **ls, cd, cat, file, du, find**

Additional Reading

- [A Surprisingly Common Mistake Involving Wildcards & The Find Command](#)
- [The Most Confusing Grep Mistakes I've Ever Made](#)
- [50 examples with sed](#)

Additional Reading: Magical first line



```
com.docker.cli
GNU nano 4.8 sless.sh
#!/bin/bash
s1="HELLO WORLD"
s2="HELLO"

if [[ "$s1" < "$s2" ]]; then
    echo "s1 is less than s2"
elif [[ "$s1" > "$s2" ]]; then
    echo "s2 is less than s1"
else
    echo "s1 and s2 are equal"
fi

^G Get Help ^O Write Out ^W Where Is ^K Cut Text ^J Justify
^X Exit ^R Read File ^\ Replace ^U Paste Text ^T To Spell
```

Magic first line that tells the system (and GNU nano) the file is a bash script

Additional Reading: Magical first line



How do you execute a script called `myscript` ?

2



```
sh myscript      # Or sh /path/to/myscript
bash myscript    # Or bash /path/to/myscript
perl myscript    # Or perl /path/to/myscript
python myscript  # Or python /path/to/myscript
```



etc.



This is easy but you do need to know which interpreter to use to run the script, and as you have noticed trying to use the wrong one will generate errors (for example using `sh` instead of `bash`).

Better, is to let the script declare the correct interpreter itself. This is defined using the `#!` on the first line. For example this line would declare `bash` (specifically `/bin/bash`) as the interpreter:

```
#!/bin/bash
```

To use this feature you need to make the script executable,

```
chmod a+x myscript
```

and then provided it is in one of the list of colon-separated directories specified in `$PATH` you can run it exactly like any other program - with just its name

```
myscript
```

However, notice that the current directory (notated as `.`) is not in `$PATH` so if you're working on a script in the current directory, you will need to provide a path to the script so that it can be found. Here the `.` represents *current directory* and it's a path relative to your current working directory (i.e. the same place):

```
./myscript      # Or /full/path/to/myscript
```

Why do we
need the magic
first line

`#!/bin/bash`?