

Lecture. 6: Shell Part VI

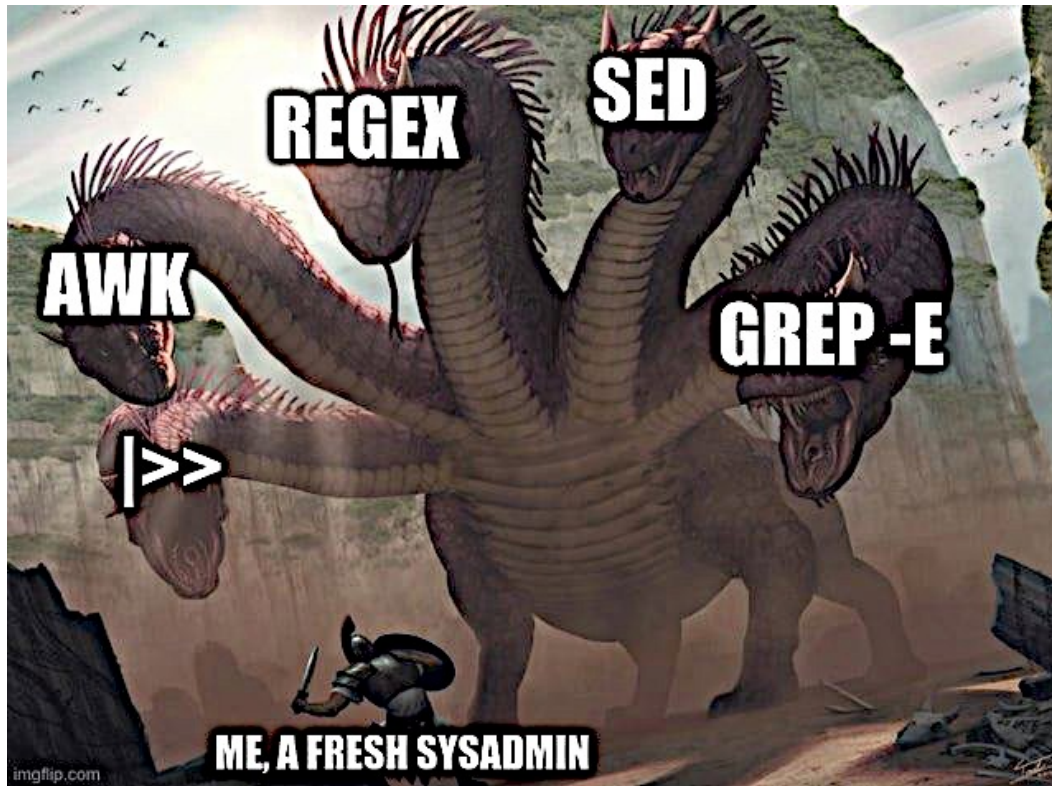
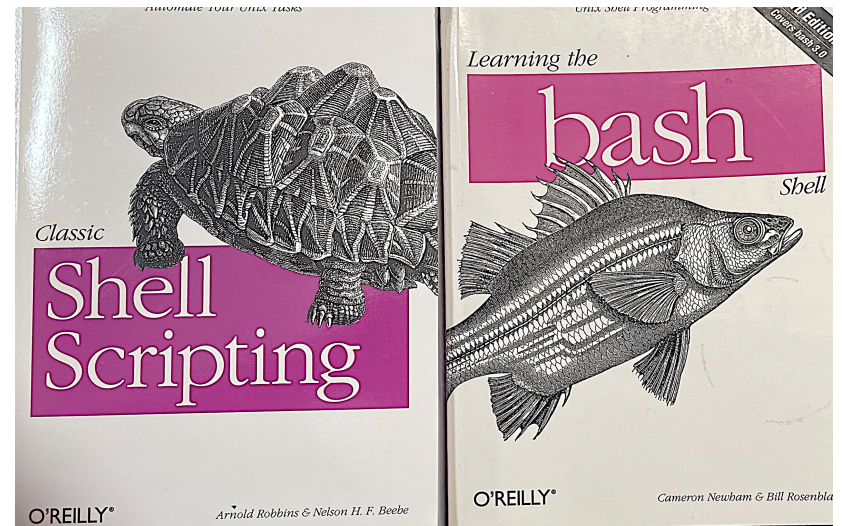


Photo of a new
Linux user learning
the command line



Suggested Literature

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



RECAP: Printing: **echo** vs **printf**

What about Colors? Works with **printf** and **echo -e**

```
whovian@phy504:~$ printf "\033[1;34m Hello World \n"
Hello World
whovian@phy504:~$ printf "\x1B[35m Hello World \n"
Hello World
whovian@phy504:~$ printf "\x1B[33m Hello World \n"
Hello World
whovian@phy504:~$ printf "\x1B[31m Hello World \n"
Hello World
whovian@phy504:~$ echo -e "\x1B[31m Hello World"
Hello World
whovian@phy504:~$
whovian@phy504:~$ echo -e "\e[31m Hello World"
Hello World
```

You need the **-e** flag on **echo** (and you don't need **\n**)

RECAP: Integer arithmetic

There is no float-point arithmetic. Integer: $\$((...))$

```
whovian@phy504:~$ echo "$(( (365 - $(date +%j) )/7 )) weeks until NYE"
50 weeks until NYE
```

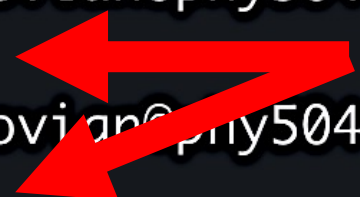
Operator	Meaning
++	Increment by one (prefix and postfix)
--	Decrement by one (prefix and postfix)
+	Plus
-	Minus
*	Multiplication
/	Division (with truncation)
%	Remainder
**	Exponentiation ^[10]

```
whovian@phy504:~$ date
Sun 15 Jan 2023 09:23:48 PM UTC
whovian@phy504:~$ date +%j
015
```

Increment by one

Because of the use of $\$(())$, we could have defined $x=2$ without declare

```
whovian@phy504:~$ declare -i x=2
whovian@phy504:~$ echo $x
2
whovian@phy504:~$ echo $((x++))
2
whovian@phy504:~$ echo $((++x))
4
```



RECAP: Comparing Strings

How can we know if two strings are the same (case sensitive)

```
GNU nano 4.8
#!/bin/bash
s1="HELLO WORLD"
s2="Hello World"
if [ "$s1" = "$s2" ]; then
    echo "s1 and s2 are the same"
else
    echo "Not Same"
fi
```

Question with
T/F possible
answers

This runs if
True

This is the first time
we also introduce
if/else statements

Be careful

a = 2

is assignment while

"\$a" = "\$b"

can be string
comparison

```
whovian@phy504:~/host$ bash sequa1.sh
Not Same
```

HW: use the operator **!=** which returns T if strings are not equal

RECAP: if/elif/else statements

if / elif / else with file attribute checking (super useful)

Operator	True if...
----------	------------

-a <i>file</i>	<i>file</i> exists
----------------	--------------------

-d <i>file</i>	<i>file</i> exists and is a directory
----------------	---------------------------------------

-e <i>file</i>	<i>file</i> exists; same as -a
----------------	--------------------------------

-f <i>file</i>	<i>file</i> exists and is a regular file (i.e., not a directory or other special type of file)
----------------	--

-r <i>file</i>	You have read permission on <i>file</i>
----------------	---

-s <i>file</i>	<i>file</i> exists and is not empty	Incomplete list
----------------	-------------------------------------	-----------------

-w <i>file</i>	You have write permission on <i>file</i>
----------------	--

```
whovian@phy504:~/host$ bash ifelse2.sh nofile.sh
file nofile.sh does not exist.
```

```
#!/bin/bash
if [ ! -e "$1" ]; then
    echo "file $1 does not exist."
    exit 1
fi
```

! opposite positional parameters

Built-in positional parameters. They have names 1, 2, 3...

RECAP: Loops: arithmetic **for** loops

The form of an arithmetic for loop is similar to those found in C/C++

```
for (( initialisation ; ending condition ; update ))  
do  
    statements...  
done
```

```
GNU nano 4.8  
#!/bin/bash  
  
for (( i=1; i <= 5; i++ ))  
do  
    for (( j=1; j <=5; j++ ))  
    do  
        echo -ne "$(( j*i ))\t"  
    done  
  
    echo  
done
```

Arithmetic for loop useful when
dealing with arrays

```
whovian@phy504:~/host$ bash forloop.sh  
1      2      3      4      5  
2      4      6      8      10  
3      6      9      12     15  
4      8      12     16     20  
5      10     15     20     25
```

RECAP: Loops: **for loops** (iteration)

You can loop over files in a directory recursively using find

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

```
x=$(find . -type f)
```

command
substitution



```
for file in $x; do
```

```
    if [ -f "$file" ]; then
```

```
        echo "$file is a regular file"
```

```
    fi
```

```
done
```

```
echo -e "\n\nAnother way to write the same loop\n\n"
```

```
for file in $(find . -type f); do
```

```
    if [ -f "$file" ]; then
```

```
        echo "$file is a regular file"
```

```
    fi
```

```
done
```

RECAP: Bash Functions

Functions can be considered a box

Input comes in, and output goes out.

How many inputs? Depends on the function.

```
whovian@phy504:~$ say_hello() {  
    echo "Hello, $1!"  
}
```

```
whovian@phy504:~$ calculate_square() {  
    local number=$1  
    local result=$((number * number))  
    echo $result  
}
```

```
whovian@phy504:~$ greet() {  
    local name=${1:-"World"} # Default value if no argument provided  
    echo "Hello, $name!"  
}
```

```
whovian@phy504:~$ say_hello  
Hello, !
```

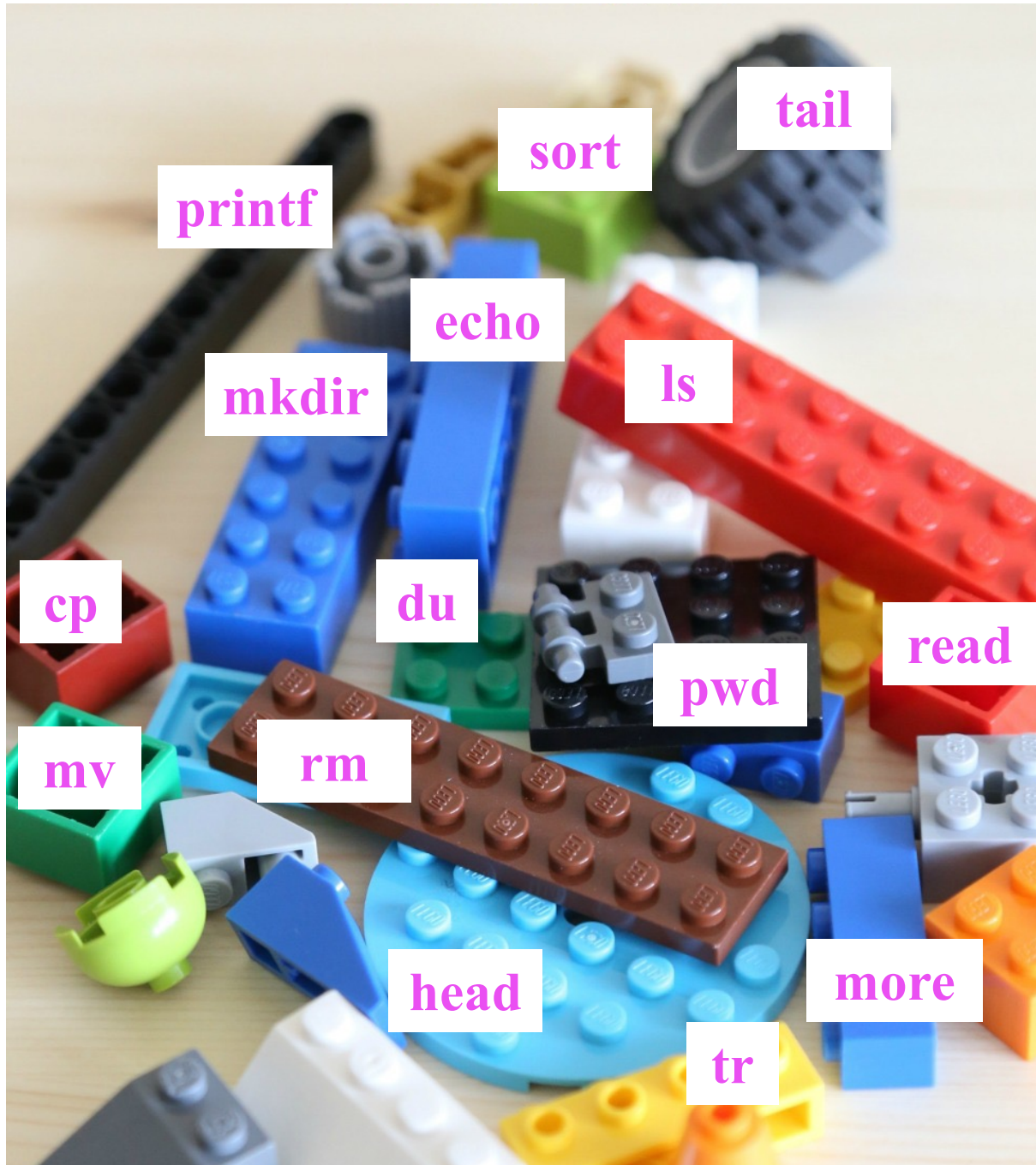
```
whovian@phy504:~$ calculate_square 3  
9
```

```
whovian@phy504:~$ greet Vivian  
Hello, Vivian!
```

local (required
keyword) variable:
defined only inside
the function

Input positional parameters

Piping: bash scripts is like playing Lego



We have many small pieces (programs).

From “gluing” of programs we build pipelines



```
whovian@phy504:~/host$ more singers.txt
4 Avril_Lavigne.
10 Rihanna.
2 Lady_Gaga.
5 Usher
14 Shakira
1 Justin_Timberlake.
20 Billie_Joe_Armstrong
3 Taylor_Swift.
whovian@phy504:~/host$ sort -n singers.txt
1 Justin_Timberlake.
2 Lady_Gaga.
3 Taylor_Swift.
4 Avril_Lavigne.
5 Usher
10 Rihanna.
14 Shakira
20 Billie_Joe_Armstrong
whovian@phy504:~/host$ sort -nr singers.txt
20 Billie_Joe_Armstrong
14 Shakira
10 Rihanna.
5 Usher
4 Avril_Lavigne.
3 Taylor_Swift.
2 Lady_Gaga.
1 Justin_Timberlake.
```

Sorting integers with **sort**

File with singers and the
number of albums you own
from these singers

- **-n** option: interpret
column as numbers (not as
strings)
- **-r** option: reverse order

What if the numbers we want
to sort are not in the first
order

```
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
```

Sorting integers with **sort**

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

```
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`

Sorting
dictionary
ordering

Sorting
using
multiple
columns

```
whovian@phy504:~/host$ more sorting3.txt
```

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

```
whovian@phy504:~/host$ sort -t"," --key 1,1d --key 3,3g sorting3.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
```



Sorting
with
different
separators

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!

```
whovian@phy504:~$ more stablesort.txt
```

```
1 4
1 3
1 2
1 1
2 1
2 2
2 3
```

```
whovian@phy504:~$ sort stablesort.txt -s -k1,1
```

```
1 4
1 3
1 2
1 1
2 1
2 2
2 3
```

```
whovian@phy504:~$ sort stablesort.txt -k1,1
```

```
1 1
1 2
1 3
1 4
2 1
2 2
2 3
```

Stable Sorting

Stable sort algorithms sort equal elements in the same order that they appear in the input.

Piping commands

Example: piping **sort** and **head** commands with |

```
whovian@phy504:~/host$ sort -n singers.txt
```

```
1 Justin_Timberlake.
```

```
2 Lady_Gaga.
```

```
3 Taylor_Swift.
```

```
4 Avril_Lavigne.
```

```
5 Usher
```

```
10 Rihanna.
```

```
14 Shakira
```

```
20 Billie_Joe_Armstrong
```

```
whovian@phy504:~/host$ head -n 3 singers.txt
```

```
4 Avril_Lavigne.
```

```
10 Rihanna.
```

```
2 Lady_Gaga.
```

```
whovian@phy504:~/host$ sort -n singers.txt | head -n 3
```

```
1 Justin_Timberlake.
```

```
2 Lady_Gaga.
```

```
3 Taylor_Swift.
```

Lego analogy starts to
make sense now
building blocks



Removing duplicate words

Piping is powerful: lets pip **sort** and **uniq** command

```
whovian@phy504:~/host$ more portugues-numbers
```

```
quatro  
um  
seis  
sete  
dois  
dez  
cinco  
tres  
nove  
dez  
quatro  
oito  
oito  
oito
```

Portuguese
numbers between
one and ten with
repetition

uniq command **filters**

adjacent lines from input

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

Because of the word adjacent,
we to first sort the lines

```
whovian@phy504:~/host$ sort portugues-numbers | uniq -u
```

```
cinco  
dois  
nove  
seis  
sete  
tres  
um
```

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
```

Counting lines, words and chars

wc command counts bytes/chars/words or lines

- **-c** count bytes
- **-w** count words
- **-l** count lines

```
whovian@phy504:~/host$ echo This is a test of the emergency broadcast system | wc -l
1
whovian@phy504:~/host$ echo This is a test of the emergency broadcast system | wc -w
9
whovian@phy504:~/host$ echo This is a test of the emergency broadcast system | wc -c
48
whovian@phy504:~/host$ echo This is a test of the emergency broadcast system | wc
    1    9   48
```

wc accepts one or multiple files as input – producing one-line reports for each, followed by summary report

```
whovian@phy504:~/host$ wc /etc/passwd /etc/group
 25   35 1324 /etc/passwd
 45   45  569 /etc/group
 70   80 1893 total
```

Reformatting file output with **pr**

pr command paginate or columnate file(s) for printing

Few Options: **-d** double space **-wNUM** width **-l** count lines

```
whovian@phy504:~/host$ more table.txt | pr --columns=3 -t -w80
2      3      4      5 12 18      24      30 22 33      44      55
4      6      8      10 14 21      28      35 24 36      48      60
6      9      12     15 16 24      32      40 26 39      52      65
8      12     16     20 18 27      36      45 28 42      56      70
10     15     20     25 20 30      40      50 30 45      60      75
whovian@phy504:~/host$ more table.txt | pr --columns=3 -t -w100
2      3      4      5      12      18      24      30      22      33      44      55
4      6      8      10     14      21      28      35      24      36      48      60
6      9      12     15     16      24      32      40      26      39      52      65
8      12     16     20     18      27      36      45      28      42      56      70
10     15     20     25     20      30      40      50      30      45      60      75
whovian@phy504:~/host$ more table.txt | pr --columns=3 -t -w100 -d
2      3      4      5      12      18      24      30      22      33      44      55

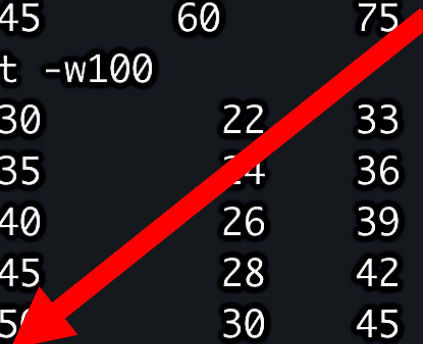
4      6      8      10     14      21      28      35      24      36      48      60

6      9      12     15     16      24      32      40      26      39      52      65

8      12     16     20     18      27      36      45      28      42      56      70

10     15     20     25     20      30      40      50      30      45      60      75
```

omit header & footer notes



Removing CAPS in a text

tr SET1 [SET2] is a powerful command that translate, squeeze, and/or delete chars from standard input, writing to standard output.

- **-c** use the complement of **SET1**
- **-d** delete characters in **SET1**, do not translate
- **-s** replace sequence of a repeated chars with a single occurrence
- **-t** first truncate **SET1** to length of **SET2**

```
whovian@phy504:~/host$ echo Vivian viVIAN viViAN VIVIAN | tr A-Z a-z  
vivian vivian vivian vivian
```

Replacing nonletters (ex: spaces) with newline

```
whovian@phy504:~/host$ echo Vivian viVIAN viViAN VIVIAN | tr -cs A-Za-z\' '\n'  
Vivian  
viVIAN  
viViAN  
VIVIAN
```

Application 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 (sed \$1 q = print first \$1 lines)

Application 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
```

Application counting words on Hamlet

How many unique words? (uses regex)

```
whovian@phy504:~/host$ bash wf.sh 999999 < hamlet.txt | grep -c '^ *1.'  
2847  
whovian@phy504:~/host$
```

Regex (regular expressions) is the next topic using a power editor called **sed**

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
```

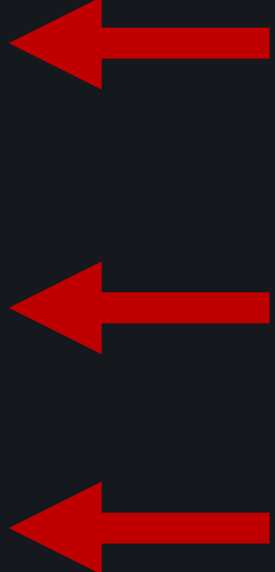
Nice documentation: <https://www.gnu.org/software/sed/manual/>

sed – string substitution

sed for string substitution:



```
whovian@phy504:~/host$ sed '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 Massachusetts
```



- note the / (3 of them all required) ‘s/oldtext/newtext/’
- Note that sed wrote the altered text in the terminal. To do in-place modifications we need to use the **-i** option

sed – string substitution

what about multiple substitutions? Use -e



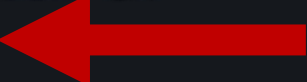
```
wovian@phy504:~/host$ sed -e 's/ MA/, Massachusetts/'  
-e 's/ PA/, Pennsylvania/' address_list.txt  
John Daggett, 341 King Road, Plymouth, Massachusetts  
Alice Ford, 22 East Broadway, Richmond VA  
Orville Thomas, 11345 Oak Bridge Road, Tulsa OK  
Terry Kalkas, 402 Lans Road, Beaver Falls, Pennsylvania  
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 the space before MA. Be careful with characters between //
- Output redirection to save to a new file `sed -f sedscr list > newlist`

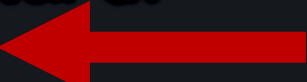
sed – string substitution

You can limit the lines where sed can replace strings

```
whovian@phy504:~/host$ sed -e '1,4s/ 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 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$ sed -e '1,5s/ 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 – string substitution

sed can replace only lines that contains 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 – string substitution

For longer scripts – create a file and use -f

```
GNU nano 4.8                                sedscript
s/ MA/, Massachusetts/
s/ PA/, Pennsylvania/
s/ CA/, California/
s/ VA/, Virginia/
s/ OK/, Oklahoma/
```



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

Homework I (option processing)

- Write a program with two options. `-f` option makes the program write a positive message (e.g., “The World is Wonderful”). `-g` option makes the program write a negative message (e.g., “The World is Terrible”). All other letters (e.g., `-g` or `-z`), makes the program write a neutral message (e.g., “The world is average”)

Homework II

- Explain what this function does. Code it yourself and run an example. What is the most surprising or challenging bash feature of this code?

```
backup() {  
    local file="$1"  
    local backup="${file}.${date +%Y%m%d_%H%M%S}.bak"  
    cp -a "$file" "$backup" && echo "Created backup: $backup"  
}
```

- Modify **backup()** to work with folders. This includes researching the **tar** compressing tool to learn how to compress a folder to a ***.tar** file.
- The most efficient lossless compressing tool in linux is **xz**. The only problem is that **xz** does not compress folders. But with piping, we can overcome that limitation. Adapt the example below to create a **backupxz()** function that can backup files and directories with xz.

```
whovian@phy504:~$ ls  
2501.16800 2501.16801 2501.16802 bash.bashrc host script2.sh script.s script.sh test1.txt test2.txt  
whovian@phy504:~$ mkdir test  
whovian@phy504:~$ mv 2501.* ./test  
whovian@phy504:~$ tar -cf - test | xz -9ze -v -T2 >test.tar.xz  
100 % 17.7 MiB / 17.9 MiB = 0.985 1.4 MiB/s 0:13  
whovian@phy504:~$ ls  
bash.bashrc host script2.sh script.s script.sh test test1.txt test2.txt test.tar.xz  
whovian@phy504:~$
```

Homework III

- When I gave to **claude.ai** HW5 part III, where I asked you to create **backupxz()**, the AI solution had two interesting lines that solved two pitfalls of my solution. Try to explain what these lines do.

```
backupxz() {  
    local source="$1"  
    local timestamp=$(date +%Y%m%d_%H%M%S)  
  
    (...)  
  
    source="${source%}/" # line 1  
    local basename=$(basename "$source") # line 2  
    local backup="${basename}_${timestamp}.tar.xz"  
  
    (...)  
}
```

- In all these examples, we used the bash command

date +%Y%m%d_%H%M%S

Research its syntax and explain its functioning in plain words (including the crypt **%Y....%S** argument.

Homework IV

- Write a script that takes a string as input and checks if it is a palindrome (i.e., a word that is spelled the same way forwards as it is backwards). The script should output "String is a palindrome" if the string is a palindrome, or "String is not a palindrome" if it is not
- Write a script that extracts the second and third field of a file and sort the output numerically
- Write a script that extracts the first field of a file (table) and only print the unique values
- Write a script that extracts the characters from the 11th to the 20th of each line in a file, but only prints lines that contain the string "example" (hint: use and research `grep`)
- Write a script that displays the top 10 largest files in a directory (hint: use and research the command `du`)
- Write a script that counts the number of files in a directory (hint: use and research the command `find` with command-line option `-type`)

Homework V

- The command

```
tr -dc 'a-zA-Z0-9~!@#%^&* _()+}{?></";.,[]=-' < /dev/urandom | fold -w 32 | head -n 1
```

generates 32 random characters.

Try it yourself and explain how this program works
(as much as you can)

```
whovian@phy504:~$ tr -dc 'a-zA-Z0-9~!@#%^&* _()+}{?></";.,[]=-' < /dev/urandom | fold -w 32 | head -n 1
!dVPmtJ~^_f}rSHt^_-=%^<T_lCS+(dw
whovian@phy504:~$
whovian@phy504:~$ tr -dc 'a-zA-Z0-9~!@#%^&* _()+}{?></";.,[]=-' < /dev/urandom | fold -w 32 | head -n 1
Ad_XY~I,_dtZXN[NOEhF5.;/S*3_u)qx
whovian@phy504:~$
whovian@phy504:~$ tr -dc 'a-zA-Z0-9~!@#%^&* _()+}{?></";.,[]=-' < /dev/urandom | fold -w 32 | head -n 1
A*9~^Ii@i04J{Nm-gI0{ }gkZUw[#f>8*
whovian@phy504:~$
whovian@phy504:~$ tr -dc 'a-zA-Z0-9~!@#%^&* _()+}{?></";.,[]=-' < /dev/urandom | fold -w 32 | head -n 1
>AZ}Sa=od9x-ZS?ZHB~807]3]Sa=[kDu
whovian@phy504:~$ █
```

Homework VI (use **echo** and **tr**)

- Given the string "**Hello, World!**", write a command that converts all lowercase letters to uppercase. Hint: use both echo and tr
- Given the string "**User12345**", write a command that removes all digits, leaving only the non-digit characters. Hint: use both echo and tr
- Given the string "**This is a test.**", write a command that replaces sequences of multiple spaces with a single space.
- Given the string "**one:two:three**", write a command that replaces each colon (:) with a tab character.
- Given a multi-line string:

**"apple
Banana
Cherry"**

write a command that replaces each newline with a comma, producing:
apple,banana,cherry.

Homework VII (use **pr**)

- Consider the file **file.txt** created by **cat** write a command that formats the output in two columns.

```
whovian@phy504:~$ cat << 'EOF' > file.txt
Item1
Item2
Item3
Item4
Item5
Item6
Item7
Item8
Item9
Item10
EOF
```

The output should look like (no header)

Item1	Item6
Item2	Item7
Item3	Item8
Item4	Item9
Item5	Item10

Hint: **pr** sometimes outputs a lot of empty lines. The no header option should help. If not, squeeze them by piping the command with **tr -s**

Homework VIII (use **pr**)

- Consider the file **notes.txt** created by **cat**. Format the file with double spacing:

```
whovian@phy504:~$ cat << 'EOF' > notes.txt
Line one of notes.
Line two of notes.
Line three of notes.
Line four of notes.
Line five of notes.
Line six of notes.
Line seven of notes.
Line eight of notes.
Line nine of notes.
Line ten of notes.
Line eleven of notes.
Line twelve of notes.
EOF
```

The output should look like (no header). →

```
Line one of notes.

Line two of notes.

Line three of notes.

Line four of notes.

Line five of notes.

Line six of notes.

Line seven of notes.

Line eight of notes.

Line nine of notes.

Line ten of notes.

Line eleven of notes.

Line twelve of notes.

whovian@phy504:~$
```

Homework IX (use **pr**)

- Consider the file **data.txt** created by **cat**. Format the file into three columns with a 60-character page width:

```
whovian@phy504:~$ cat << 'EOF' > data.txt
Data1
Data2
Data3
Data4
Data5
Data6
Data7
Data8
Data9
EOF
```

The output should look like (no header).

Data1	Data4	Data7
Data2	Data5	Data8
Data3	Data6	Data9

Additional Reading

- Article about command substitution

<https://unix.stackexchange.com/a/440123/464161>

- Article about the Difference between `${}` and `$()` in Bash

<https://www.baeldung.com/linux/parameter-expansion-vs-command-substitution>