Scripting & Computer Environments

Basic Filters

IIIT-H

Aug 19, 2015

...Previously & Today...

Previously:

1. A revisit on GNU/Linux
2. Basic commands
3. File operations
   - File security

     `chmod, chgrp, chown`
   - Compression/archiving

     `tar, gzip/gunzip, bzip(2)/bunzip(2), zip/unzip...`
   - Remote file access

     `ssh, scp, sftp`
   - File editing

     `Vi/Vim` editor

Today:

- Redirection & Pipes
- Shell Wildcards
- Simple Filters

...Previously & Today...

Previously:

1. A revisit on GNU/Linux
2. Basic commands
3. File operations
    - File security

      `chmod, chgrp, chown`
    - Compression/archiving

      `tar, gzip/gunzip, bzip(2)/bunzip(2), zip/unzip...`
    - Remote file access

      `ssh, scp, sftp`
    - File editing

      `Vi/Vim` editor

Today:

- Redirection & Pipes
- Simple Filters
- Shell Wildcards

Brainstorm

1. Last class' checkpoint question?

   Assume a system with `umask` value set to 022. Create a file inside a new directory.

   - case 1: -w for the directory only
   - case 2: -w for the file only
   - case 3: -w for both

   Now, do `rm`/`mv` on the file. What happens?

2. Sources/inputs & destinations/outputs of a command?

Brainstorm

1. Last class' checkpoint question?

   Assume a system with `umask` value set to 022. Create a file inside a new directory.

   - case 1: -w for the directory only
   - case 2: -w for the file only
   - case 3: -w for both

   Now, do `rm`/`mv` on the file. What happens?

2. Sources/inputs & destinations/outputs of a command?

Brainstorm

1. Last class' checkpoint question?

   Assume a system with `umask` value set to 022. Create a file inside a new directory.

   - case 1: -w for the directory only
   - case 2: -w for the file only
   - case 3: -w for both

   Now, do `rm`/`mv` on the file. What happens?

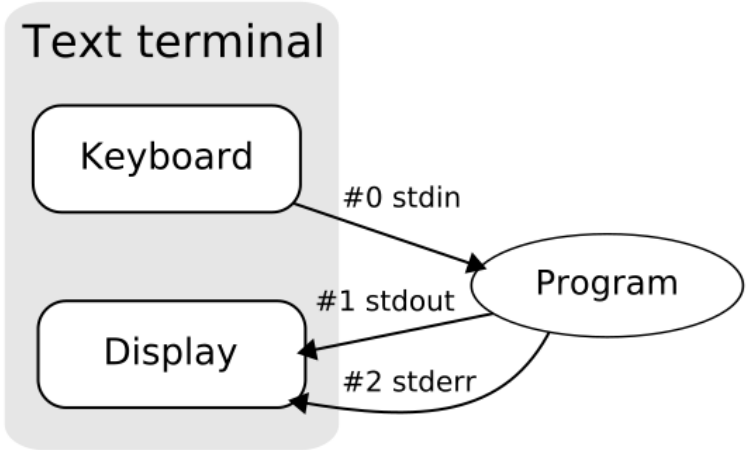
2. Sources/inputs & destinations/outputs of a command?

I/O Redirection

- Terminal?
- The shell reads input & writes output as a stream of characters. (stream = sequence of bytes)
- Command outputs: result or status/error messages. Sent to?
- The shell provides 3 special files @ login, each associated with a default terminal device.
- Each has a unique file descriptor (FD) value.
  1. #0 – Standard Input stream (STDIN): input to commands.
  2. #1 – Standard Output stream (STDOUT): output from commands.
  3. #2 – Standard error stream (STDERR): errors from commands.

I/O Redirection

- Terminal?
- The shell reads input & writes output as a stream of characters. (stream = sequence of bytes)
- Command outputs: result or status/error messages. Sent to?
- The shell provides 3 special files @ login, each associated with a default terminal device.
- Each has a unique file descriptor (FD) value.
  1. `#0 - Standard Input stream (STDIN)`: input to commands.
  2. `#1 - Standard Output stream (STDOUT)`: output from commands.
  3. `#2 - Standard error stream (STDERR)`: errors from commands.



I/O Redirection

- A way of unhooking a stream from its default device.
- Changing where input comes from/output goes to.
- The operators:
  1. Input redirection: `0< or just <`
  2. Output redirection: `1> or >, 1>> or >>`
  3. Error redirection: `2> or 2>>`

Examples

```
wc -l < /usr/share/dict/words

cat < input.txt > output.txt          (try: cat, cat << END  a.k.a. Heredoc)

cat - input.txt >> output.txt                                  (ctrl + D)

ls -l IExist.txt IExistNot.txt > output.txt 2>> log.txt

ls -l IExist.txt IExistNot.txt &> output.txt

ls -l IExist.txt IExistNot.txt 2>&1                          (??)

ls -l IExist.txt IExistNot.txt 2> log.txt 1>&2               (??)

cat /etc/passwd > /dev/null                                  (bit bucket)

{ ls -l ; echo ; cat /etc/passwd; } > output.txt              {cmd grouping}
                                                           or (cmd grouping)
```

Piping (|)

```
command1 | command2 | ...| command n
```

- Output of a command piped into input for another.
- STDOUT $\rightarrow$ STDIN
- The Shell does the setup, the command unaware.
- Length of the pipe can be "indefinite"
- Redirection vs Piping (> vs |)? E.g. No of files in ./ ?

Example

```
who | wc -l

history | head -20 | tail -5

ls -l | sort -k 8 > output.txt
```

Piping (|)

```
command1 | command2 | ...| command n
```

- Output of a command piped into input for another.
- STDOUT $\rightarrow$ STDIN
- The Shell does the setup, the command unaware.
- Length of the pipe can be "indefinite"
- Redirection vs Piping (> vs |)? E.g. No of files in ./ ?

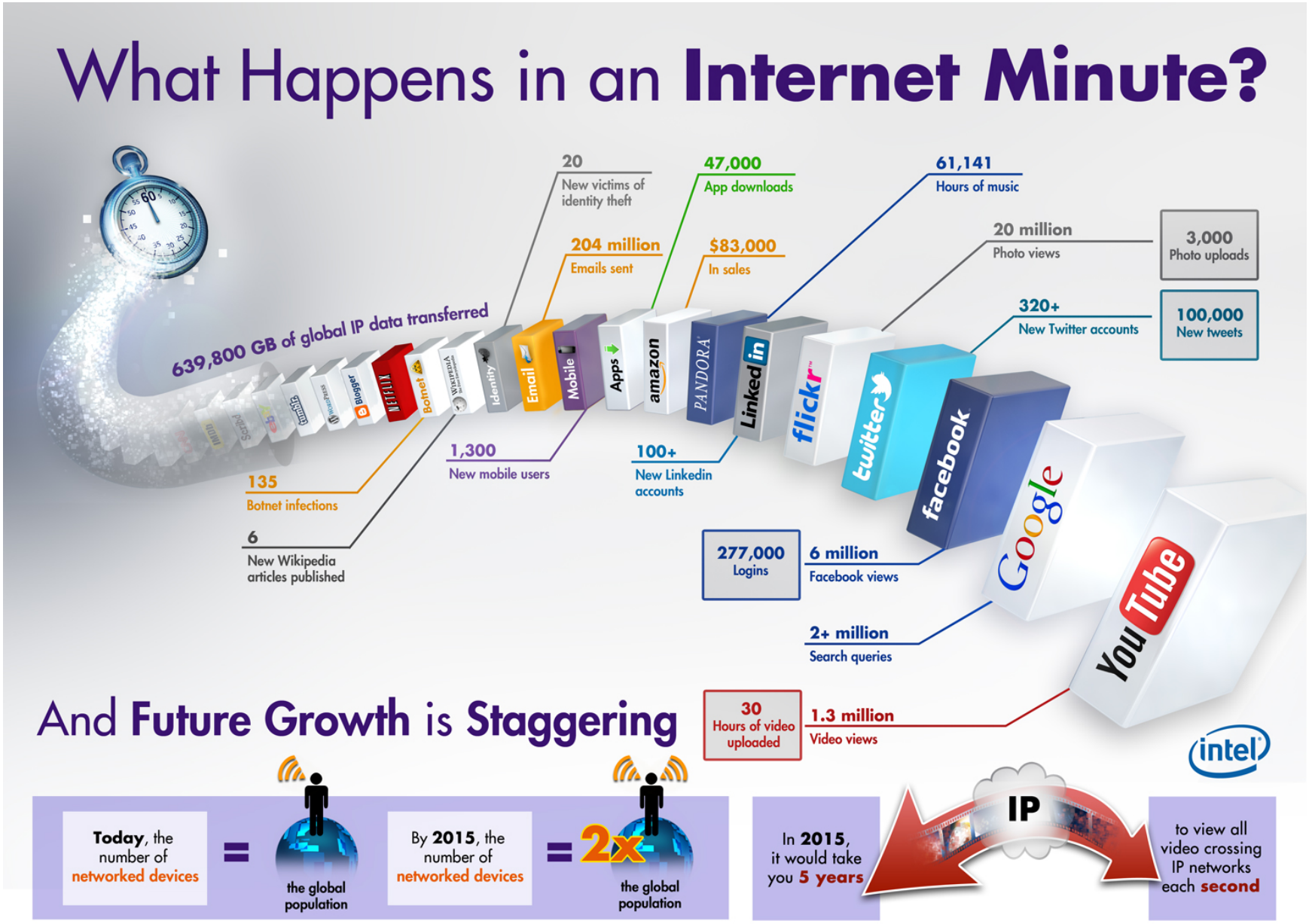
Example

```
who | wc -l

history | head -20 | tail -5

ls -l | sort -k 8 > output.txt
```

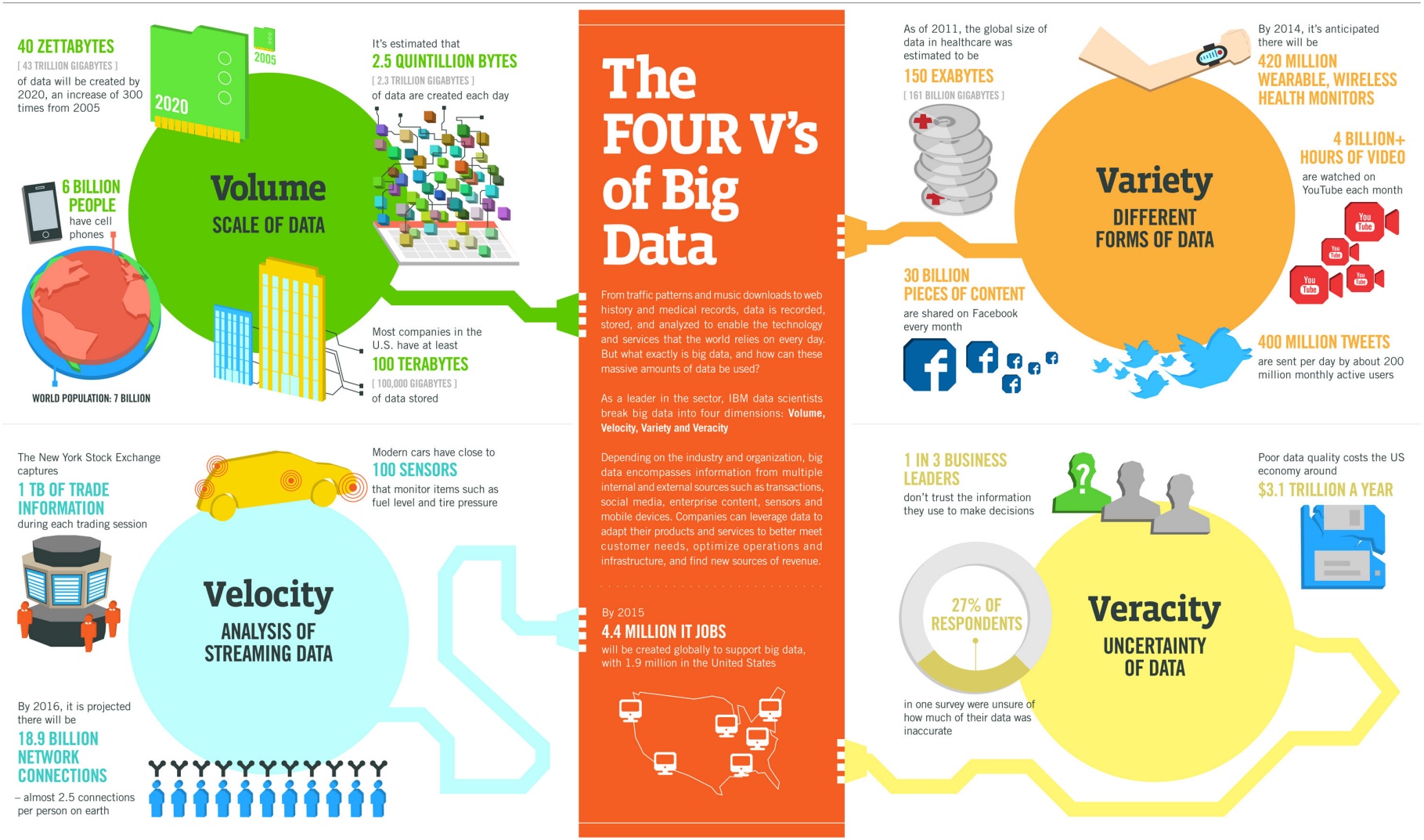
Filters

Data/Text Mining? Applications?

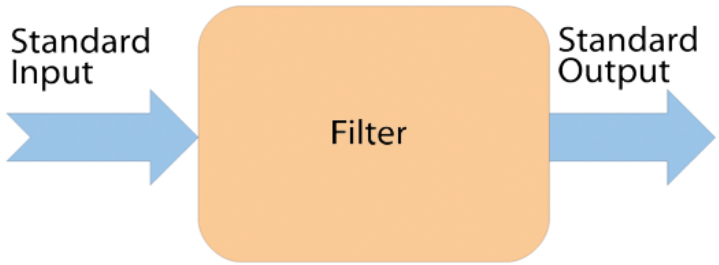
Data/Text Mining? Applications?



Big Data & Analytics



Filters



- Simply, commands that use both the `STDIN` and `STDOUT`.
- Read input stream $\rightarrow$ [transform it] $\rightarrow$ output the result.
- Example application: text filtering/processing/manipulation

  e.g. `cat`, `wc`, `tr`, `grep`, `sed`, `awk`, etc

cat (concatenate) & split

`cat [option] [file]`

- `cat` displays contents of file, if any, on `STDOUT`.
- Keeps reading from `STDIN` until `EOF` or `ctrl+d`
- `split` splits a file into pieces.

Example

```
cat file1.txt
cat file1.txt file2.txt

split -l 2 file1.txt
split -b 5 file2.txt

cat x*
```

cat (concatenate) & split

`cat [option] [file]`

- `cat` displays contents of file, if any, on `STDOUT`.
- Keeps reading from `STDIN` until `EOF` or `ctrl+d`
- `split` splits a file into pieces.

Example

```
cat file1.txt
cat file1.txt file2.txt

split -l 2 file1.txt
split -b 5 file2.txt

cat x*
```

`wc` word count

- `-l` line count
- `-w` word count
- `-m` character count

`wc` word count

- `-l` line count
- `-w` word count
- `-m` character count

sort & uniq

- Sorting by collating sequence
- Also merges already-sorted files and checks if sorted or not
- Options: `-r` (reverse order), `-k` (column), `-n` (numerical order), etc
- `uniq` discards identical lines

```
cat file1 file2 | sort | less
cat file1 file2 | sort | unique | less
```

`cmp, diff, comm` (file comparison)

- `cmp` - byte-by-byte comparison
- `comm` - line-by-line comparison of sorted files
- `diff` - which lines be changed to make them identical

`cmp, diff, comm` (file comparison)

- `cmp` - byte-by-byte comparison
- `comm` - line-by-line comparison of sorted files
- `diff` - which lines be changed to make them identical

`head` & `tail`, `cut` & `paste`

- `cut` - slits a file vertically (unlike...?)
- `paste` - merges lines of a file vertically.

e.g. Display only the permission characters of `ls -l`?

```
ls -l | cut -c1-10 > permissions.txt      (options: -d, -f,..)
paste permissions.txt file2.txt > merged.txt
```

`cmp, diff, comm` (file comparison)

- `cmp` - byte-by-byte comparison
- `comm` - line-by-line comparison of sorted files
- `diff` - which lines be changed to make them identical

`head & tail, cut & paste`

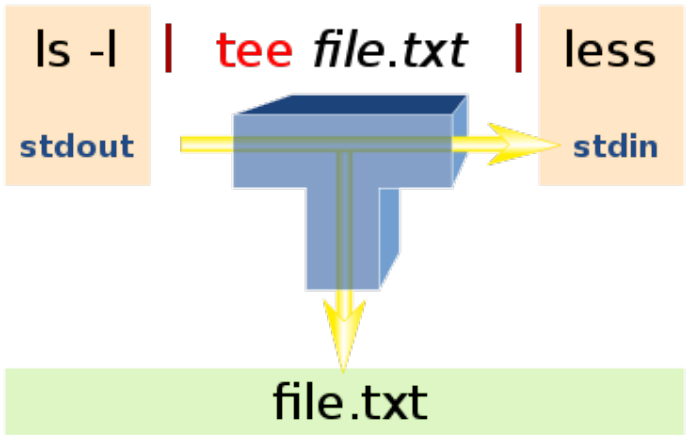
- `cut` - slits a file vertically (unlike...?)
- `paste` - merges lines of a file vertically.

e.g. Display only the permission characters of `ls -l`?

```
ls -l | cut -c1-10 > permissions.txt     (options: -d, -f...)

paste permissions.txt file2.txt > merged.txt
```

The `tee` filter

- An external command.
- Reads input from `STDIN` and duplicates it.
- One copy goes to `STDOUT`.
- Another copy goes to a file(s).




```
who | tee file.txt
```

```
history | tee -a file.txt
```

The `tr` filter

`tr [options] set1 set2`

- Translates/deletes each character in set1 to set2
- a.k.a. search and replace
- Receives input only from `stdin`. From files?

The `tr` filter

`tr [options] set1 set2`

- Translates/deletes each character in set1 to set2
- a.k.a. search and replace
- Receives input only from `stdin`. From files?

Example

```
tr ‘A-Z’ ‘a-z’

cat input.txt | tr ‘ABCD’ ‘1234’

tr ‘ ’ ‘\t’ < input.txt

echo "hello there" | tr -d ‘e’
```

Shell Wildcards

Wildcards (a.k.a. Shell Metacharacters)

- Characters with special meaning to the shell

  `*  ?  <  >  [ ]  '  "  ;  { }  ( )  !  &  ^  |  \n ...`

- Expanded by the shell first (this is known as Globbing).
- `?` matches any single character
- `*` matches 0+ number of characters (but '.' and '/')
- `[...]` matches any element in the set.
- Characters with special meaning inside `[]`: - (hyphen), $\wedge$, !
- \ turns off the special meaning

Example

```
ls -l ?????


rm -i *.c


cp [A-Z]* MyDir


ls -l file[^A-Z]*   or  ls -l file[!A-Z]*


echo \\
```

Wildcards: Searching for Files

- One of the basic operations of any OS
- Linux offers some commands: `locate`, `whereis`, `find` ...

```
find <where> -name <search criteria>
```

```
find / -name 'file[^12].c'

find ~ -name 'My??*'

find . -name '*199[0-9]*'

find . -size -2000b -mtime 1 -name '*.html'
```

Checkpoint!

1. Interpret the following command.

```
tr 'a-z' '0-9' < input.txt | sort -rn | uniq | tail > ouput.txt
```

2. Extract lines 10 through 20 of `/etc/passwd`.

3. The first 3 largest files in the current directory?

4. Remove digits from all C programs in the current directory.


RegEx
Regular Expression