sed

- Introduction
    - It is a programming language for processing text streams
    - sed is an abbreviation for **s**tream **ed**itor
    - It is a part of POSIX
    - sed precedes awk
    - use sed to pre-process input for further processing
    - sed is a meant for text processing, fast in execution
    - sed is available everywhere !
- Execution model
    - Input stream is a set of lines
    - Each line is a sequence of characters
    - Two data buffers are maintained: active **pattern** space and auxiliary **hold** space
    - For each line of input, an **execution cycle** is performed loading the line into the pattern space
    - During each cycle, all the statements in the script are executed in the sequence for matching **address pattern** for **actions** specified with the **options** provided
- usage
    - Single line at the command line
    - `sed -e 's/hello/world/g' input.txt`
    - Script interpreted by sed
    - `sed -f ./myscript.sed input.txt`
    - `myscript.sed`

    ```
    #!/usr/bin/sed -f
    2,8s/hello/world/g
    ```

- sed statements
    - `:label address pattern action options ;`
        - `address pattern`
            - `address`
            - `address, range`
            - negation `!`
        - `action`
            - Single Character action. Same as "ed" or "ex"
        - `options`
            - Depends on the `action`
- Grouping commands

    - `{ cmd; cmd; }`

- address
    - Selecting by Numbers
        - `5`
        - `$`
        - `%`
        - `1~3`
    - Selecting by matching
        - `/regexp/`
    - Range Address
        - `/regexp1/,/regexp2/`
        - `/regexp/, +4`
        - `/regexp/, ~2`
        - `5,15`
        - `5,/regexp/`

- actions

command	Description
`p`	Print the pattern space
`d`	Delete the pattern space
`s`	Substitute using regex match `s/pattern/replacement/g`
`=`	Print current input line number, \n
`#`	comment
`i`	Insert above current line
`a`	Append below current line
`c`	Change current line

- programming

command	Description
`b label`	Branch unconditionally to label
`:label`	Specify location of label for branch command
`N`	Add a new line to the pattern space and append next line of input into it.
`q`	Exit sed without processing any more commands or input lines
`t label`	Branch to label only if there was a successful substitution was made
`T label`	Branch to label only if there was no successful substitution was made

command	Description
`w filename`	Write pattern space to filename
`x`	Exchange the contents of hold and pattern spaces

- bash + sed
    - Including sed inside shell script
    - heredoc feature
    - Use with other shell scripts on command line using pipe
- Working with `sed`
    - `sed -e "" edit.txt` -The default action of sed is to just print out the contents of the file if nothing is specified.
    - `sed -n` - the default action of printing is not performed
    - `sed -e '=' sample.txt` the `=` prints the line number
    - `sed -n -e '5p' sample.txt - -n` says not to print anything else by default. `5p` says to print the 5th line. Without `-n` all the lines will be printed and the 5th line will be printed 2 times.The fifth line is the address space
    - `sed -n -e '5!p' sample.txt` The `!` means that all lines except the 5th line will be printed. `!` Exclamation mark negates an address.
    - `sed -n -e '$!p' sample.txt` prints all except the last line. Careful with using `'` instead of `"` here.
    - `sed -n -e '5,8p' sample.txt` prints the 5th to the 8th line both inclusive.
    - `sed -n -e '=; 5,8p' sample.txt` prints all line numbers and from 5 to 8 prints lines also.
    - `sed -n -e '5,8{=;p}' sample.txt` prints line numbers for the line 5 to 8 alone.
    - `sed -n -e '1~2p' sample.txt` prints lines 1,3,5,7 ... Number coming after `~` specifies step size.
    - `sed -n -e '1~2!p' sample.txt` prints the remaining lines due to the negation specified by `!`
    - `sed -n -e '/microsoft/p' sample.txt` Supplying a phrase and an action. The hrase is microsoft and the action is to print every line containing the phrase.
    - `sed -n -e '/in place of/!p' sample.txt` prints the lines that do not contain the phrase "in place of"
    - `sed -n -e '/adobe/,+2p' sample.txt` prints the line containing "adobe" and two more lines that come immediately after that.
    - `sed -n -e '5d' sample.txt` deletes the 5th line and prints the rest
    - `sed -e '5,8d' sample.txt` deletes from the 5th to the 8th line and prints the rest
    - `sed -e '1,$d' sample.txt` deletes from the 1st to the last line and prints nothing
    - `sed -e '/microsoft/d' sample.txt` deletes all the lines containing microsoft and prints the rest
    - Most popular usage of the `sed` command is to substitute one phrase with another.
    - `sed -e 's/microsoft/MICROSOFT/g' sample.txt` search and replace. `s` implies search and `g` implies global.

- `sed -e '1s/linux/LINUX/g' sample.txt` replaces 'linux' with 'LINUX' on only the first line.
- `sed -e '1,$s/in place of/in lieu of/g' sample.txt` replaces 'in place of' with 'in lieu of' from the first line to the last line.
- Modifying the incoming stream using the extended regular espression engine.
- `sed -E -e '3,6s/^L[[:digit:]]+ //g' sample.txt` performs a search and replace from the 3rd to the 6th line of capital L followed by number/s and then a space. `-E` indicates that the Extended regular expression set should be used.
- `sed -E -e '3,/symbolic/s/^L[[:digit:]]+ //g' sample.txt` performs a search and replace from the 3rd to the line where the phrase 'symbolic' occurs, of capital L followed by number/s and then a space. `-E` indicates that the Extended regular expression set should be used.
- `sed -E -e '1~3s/^L[[:digit:]]+ //g' sample.txt` performs a search and replace from the 1st line every third line, of capital L followed by number/s and then a space. `-E` indicates that the Extended regular expression set should be used.
- `sed -E -e '1~3!s/^L[[:digit:]]+ //g' sample.txt` Negation of the address range performs the opposite of the previous command.
- Address range as a regular expression
- `sed -E -e '/text/,/video/s/^L[[:digit:]]+ //g' sample.txt` performs a search and replace from the line that contains 'text' to the line where the phrase 'video' occurs, of capital L followed by number/s and then a space. `-E` indicates that the Extended regular expression set should be used.
- `sed -e '1i ----------------header----------' -e '$a -----------footer---------' sample.txt` Here `i` inserts before the first line and `a` appends after the last line.
- `sed -e '/microsoft/i ----------------watchout----------' -e '/in place of/a -----------alternative---------' sample.txt` 'watchout' appears above every line that contains microsoft and alternative comes below every line that contains 'in place of'
- `sed -e '1~5i ----------------break----------' sample.txt` inserts 'break' after every 5 lines.
- `sed -e '/microsoft/c ------censored--------' sample.txt` For every line that has 'microsoft' `c` or change command is executed
- `sed -e '1~3c ------censored--------' sample.txt`

- An sed script file

  - `more hf.sed`

```
#!/usr/bin/sed -f
1i ------header--------
$a ------footer--------
1,5s/in place of/in lieu of/g
6i ------ simpler stuff here onward --------
6,$s/in place of.*//g
```

- First line mentions the interpreter
- last line removes all the characters whenever 'in place of' is encountered

- `sed -f hf.sed sample.txt` The `-f` implies that sed will use a file.
- `more clean.sed`

```
/[[:alpha:]]{2}[[:digit:]]{2}[[:alpha:]][[:digit:]]+/!d
s/[ ]+/ /g
s/ ([[:digit:]]+).*/ \1/g
```

- For the input file block-ex-6.input - File containing roll number and fees paid
- First line deletes all lines that dont contain roll number
- 2nd line replaces multiple spaces with single space
- 3rd line keeps number by back referencing
- `sed -E -f clean.sed block-ex-6.input`
- Joining lines
  - Example : joining lines which are ending with a `\`
  - `cat join.sed`

```
#!/usr/bin/sed -f
:x /\\$/N
/\\/s/\\\n//g
/\\$/bx
```

- The `:` indicates a label. Whenever there is a `\` the `N` causes it to read the next line in the buffer.
- 2nd line - On the lines which have `\`, if there is a new line character it will be replaced with null.
- 3rd line on those lines which contain `\` we branch to the first line.
- `sed --debug -f join.sed sample-split.txt`
- The debug option helps to debug infinite loops in sed

L7.2

Version Control

- Every Save is effectively a new version of the code
  - "Make" - Compile only those parts of code that has changed. You do not touch what has not been modified.

    - If a group of programmers are working on a project with lots of codes and lots of files, following a modlar approach (Each function as a separate file in C for example). There is a tacit understanding that programmers are not going to work on the same file.
    - Each programmer has multiple verions of the each file they worked on.
    - Why is version control necessary ? To trace back to a woring version of code.
    - Versions will depend on number of users, number of files and number of versions. This needs to be kept in a database.
    - Two major version control systems
        - SVN - Centrally hosted and managed version system
            - Allows for one master who keeps track of the version of code that is being officially supported.
            - Storage Systems - Not if it fails but when it fails - When it fails no one can access. RAID - Redundant Array of Inexpensive/Independent Disks.
        - GIT - Distributed version control system
            - Even if something happens to the master server disappears nothing significant is lost because every collaborator has a copy of everything.
            - GIT system doesn't really require a server

- git

    - remote - server with which we synchronize
    - protocol for connection - git protocol - protocol by which we exchange information with remote and do version control.
    - options of using git
        - locally run git server
        - campus git server
        - gitlab
        - github.com
    - Two factor authentication for github
        - app -> otp -> enter
        - app -> ask -> swipe
        - SMS -> OTP -> enter
        - customised for each repository/activity
        - personal access token

- Activities

    - register on github.com
    - enable 2 factor authentication (Microsoft Authenticator App)
    - Create a repo
    - practice how to pull,push,git actions.
    - Developer Settings >> Personal Access Token
    - Create repository
    - `git clone url-of-github-rep.git`
        - append `.git` to url
        - folder will be created automatically

- edit README.md using `vi README.md`
- `git init` in the directory so that git understands that it is the same directory
  - creates a `.git` folder with all the paraphernelia that git requires
  - `ctrl + z` puts the program that was running as a background job. `kill %` kills the background job
- `git remote add master url-of-github-rep`
  - it understands that there is a remote location that you hae configured
- `git config --global user.name "your_username_on_github"`
- `git config --global user.email "your_github_registered_email"`
- `git status` will chec what is happening
- `git add README.md`
- `git commit -m "Message which is describes what you have done"`
- Use the personal access token created earlier
- `vi ../pat` to store the token one level above the folder.
- `git push` enter username and personal access token

L7.3

Github Brief Introduction

- Create account on Github
  - Configure 2 factor authentication and download recovery keys
  - Install Microsoft Authenticator App on your mobile
  - Login to github.com using TFA as a habit
- Creating your own repository
  - Create private repository
  - Get a personal access token to use this
  - Clone the repository on your computer `git clone url_of_the_repo`
  - Configure the folder for git using `git init`
  - Tell git about yourself : `git config`
    - `git config --global user.name "your_username_on_github"`
    - `git config --global user.email "your_github_registered_email"`
  - Configure the remote `git remote add master url_of_the_repo`
  - Change some files if you wish
  - Run the `git status` command to understand what is going on.
  - Stage them to be ready to send to remote using `git add modified_filename`
  - `git add .` will push everything to the server
  - Commit the change using `git commit -m "message"`
  - Type `git status` again
  - Push the changes using `git push`

- Working with branches

  - Create a new branch for a repository you are already working on - `git branch` - `git branch "Panda"`
  - Check out the branch
    - `git checkout`
    - `git checkout Panda`
    - `git status` shows that you are on the Panda branch
  - Make some changes to some files
  - All changes are now to the branch
    - `git add README.md`
    - `git commit -m "This is from my PC"`
    - `git push --set-upstream origin Panda`
  - Merge the branch with the master/main
    - `git checkout main`
    - `git merge Panda`
  - On the website 'Compare and Pull Request'. Then 'Create Pull Request'. Then 'Merge Pull Request'
  - Remote checks if there is any confilct. Creating and merging branches is part of the coding cycle.

- Contributing to others' repositories

  - Fork their repository
  - create your branch
  - make some changes to your branch and push those to the server
  - on the remote server, compare and create a pull request

- Allowing contributors to chip in

  - Look at pull requests and approve them
  - Resolve and conflicts in some files