Week 4 Notes

Software Management

- Using Package Management Systems
    - Tools for installing, updating, removing and managing software
    - Install new / updated software across network
    - Package - File look up, both ways
        - Which files are given by a particular package and which package contains a given file
    - Database of packages on the system including versions (compatibility and requirements)
    - Dependency checking
    - Signature verification tools (to check authenticity of source of the software)
    - Tools for building packages (to build packages from soure code - particularly true for kernel modules)
- Package types
    - Package
        - RPM
            - Red Hat
                - CentOS
                - Fedora
                - Oracle Linux
            - SUSE Enterprise Linux
                - OpenSUSE
        - DEB
            - Debian
                - Ubuntu
                    - Mint
                - Knoppix
- Commands
    - `lsb_release -a` to find version of Operating System
    - When searching for packages for this version of the OS you can search by OS code name eg: `focal`
- Architectures
    - amd64 | x86_64
    - i386 | x86
    - arm (RISC5 Sakthi)
    - ppc64el | OpenPOWER
    - all | noarch |src (not tied to any architecture)
- Commands

- `uname -a` gives the kernel version and the type of architecture.

- Tools
  - Package Type
    - RPM
      - Yellowdog Updater Modifier (yum)
        - Red Hat Package Manager (rpm)
        - Dandified YUM (dnf)
    - DEB
      - synaptic (GUI)
      - aptitude (Command Line)
        - Advanced Package Tool (apt)
          - dpkg
            - dpkg-deb

- Package managemet in Ubuntu using `apt`
  - Inquiring package db
    - Search packages for a keyword
      - `apt-cache search keyword`
    - List all packages
      - `apt-cache pkgnames`
      - `apt-cache pkgnames | sort | less` for page by page sorted display
      - `apt-cache pkgnames nm` for all packages starting with nm
    - Display package records of a package
      - `apt-cache show -a package`

- Package Names
  - Package
    - RPM
      - package-version-release.architecture.rpm
    - DEB
      - package_version-revision_architecture.deb
      - eg : `pool/universe/n/nmap/nmap_7.80+dfsg1-2build1_amd64.deb`

- Package Priorities
  - required : essential to proper functioning of the system
  - important : provides functionality that enables the system to run well
  - standard : included in a standard system installation
  - optional : can omit if you do not have enough storage
  - extra : could conflict with packages with higher priority, has specialized requirements, install only if needed.
  - Priority is displayed as `extra` in the output of `apt-cache show nmap` or `apt-cache show wget` for example.

- Package Sections

    - Package Sections for Ubuntu focal
    - `apt-cache show fortunes` shows `Section : universe/games`

- Checksums
    - For a small change in the original file the checksum is very different. This is useful to chack if the original file has been tampered or not.
    - Can be used to verify that nothing has gone wrong to the contents of the file while downloading.
    - md5sum
        - 128 bit string
        - `md5sum filename`
    - SHA1
        - 160 bit string
        - `sha1sum filename`
    - SHA256
        - 256 bit string
        - `sha256sum filename`

4.2

- Who can install packages in Linux OS ?
    - administrators
    - sudoers in the case of Ubuntu
    - Only sudoers can install/upgrade/remove packages
    - a sudo command can be executed by those who are listed in `/etc/sudoers`
    - Command `sudo cat /etc/sudoers` . If the current `$USER` is not in the sudoers file the incident will be reported.
    - In the file the users listed under `# User privilege specification` have sudo permission.
    - sudo attempts and authentication failures get recorded in `/var/log/auth.log` . View using `sudo tail -n 100 /var/log/auth.log`
- When installing a package the system knows the website/server from which the packages have to be downloaded
    - This information is stored in the folder `/etc/apt`
    - Uncommented lines in the file `sources.list` have the debian/ubuntu sources
    - A directory `sources.list.d` stores sources for third party software. Allows `apt update` to know new versions to download from repositories stored in these files
    - Synchronize package overview files - `sudo apt-get update` fetches updates and keeps them in cache
    - Upgrade all installed packages - `sudo apt-get upgrade` upgrades the packages. It lists how many updates are going to be affected and how much data is going to be downloaded.
    - `sudo apt autoremove` to remove unused packages that were earlier installed to satisfy a particular dependency but are not needed now.
    - Install a package - `sudo apt-get install packagename`

    - `sudo apt-get remove packagename` to remove a particular package
    - `sudo apt-get reinstall packagename` to fix problems caused by accedential file deletions.
    - Clean local repository of retreived package files - `apt-get clean`
    - Purge package files from the system - `apt-get purge package`
- Package management in Ubuntu using `dpkg`
    - Allows installation directly from a `.deb` file. Package management at a lower level.
    - `/var/lib/dpkg` has some information about the packages
        - Files - `arch` , `available` , `status`
            - `cat arch` displays the architectures for which packages have been installed on the system - `amd64` , `i386`
            - `less available` displays list of packages with info.
            - `less status` displays if a particular package is installed or not
        - Folder - `info`
            - contains a set of files for each of the packages that have been installed
            - `ls wget*` will give files with information about `wget`
                - `more wget.conffiles` gives location of configuration file
                - `more wget.list` displays list of files that would get installed on the system with the package
                - `more wget.md4sums` displays the listof md5sums of the installed files. (Used to catch tampering)
- Using `dpkg`
    - List all packages whose names match the pattern
        - `dpkg -l pattern`
    - List installed files that came from packages
        - `dpkg -L package`
    - Display/Report the status of packages
        - `dpkg -s package`
    - Search installed packages for a file
        - `dpkg -S pattern`
        - eg : `dpkg -S /usr/bin/perl` shows the package from which the executable has come. ie : `perl-base`
    - To query the dpkg database about all the packages - `dpkg-query`
        - Example `dpkg-query -W -f='${Section} ${binary:Package}\n' | sort | less`
        - Example where output is filtered `dpkg-query -W -f='${Section} ${binary:Package}\n' | grep shells`
- Installing a deb package
    - `dpkg -i package_version-revision_architecture.deb`
    - not a good idea since it may have some dependencies that will have to be taken care of manually
    - Do not download deb files from unknown sources and install it on the system
    - By default use package management pointing to a reliable repository
    - Uninstalling packages using `dpkg` is NOT recommended. You may be removing a package that is required by many other packages.
- When compatibility issues cannot be resloved one can use `snap` or `docker` as alternatives when you are unable to install a particular version of a package.

4.3

Pattern Matching

- Regular Expressions `regex` and `grep` commands
  - POSIX standard
    - IEEE 1003.1-2001 IEEE Standard for IEEE Information Technology – Portable Operating System Interface (POSIX(TM))
    - Refer
  - POSIX defines regular expressions to be of 2 different types - Basic and Extended.
- Regex
  - regex is a pattern template to filter text
  - BRE: POSIX Basic Regular Expression engine
  - ERE: POSIX Extended Regular Expression engine
- Why learn regex?
  - PRocess some input from the user or perform some string operations.
  - Languages: Java, Perl, Python, Ruby, ...
  - Tools: grep, sed, awk, ...
  - Applications: MySQL, PostgreSQL, ...
- Usage
  - `grep 'pattern' filename` - to operate on every line in the file
  - `command | grep 'pattern'`
    - the `grep` command operates line after line. A common feature in many utilities in linux.
    - enclose pattern in single quotes
  - Default engine: BRE
  - Switch to use ERE in 2 ways:
    - `egrep 'pattern' filename`
    - `grep -E 'pattern' filename`

Special characters (BRE & ERE)

Character	Description
.	Any single character except null or newline
*	Zero or more of the preceding character / expression
[]	Any of the enclosed characters; hyphen (-) indicates character range
^	Anchor for beginning of line or negation of enclosed characters
$	Anchor for end of line
\	Escape special characters

Special characters (BRE)

Character	Description
`\{n,m\}`	Range of occurances of preceding pattern at least n and utmost m times
`\( \)`	Grouping of regular expressions

Special characters (ERE)

Character	Description
`{n,m}`	Range of occurances of preceding pattern at least n and utmost m times
`()`	Grouping of regular expressions
`+`	One or more of preceding character / expression
`?`	Zero or one of preceding character / expression
`|`	Logical OR over the patterns

Character Classes

Class	Description
`[[:print:]]`	Printable
`[[:alnum:]]`	Alphanumeric
`[[:alpha:]]`	Alphabetic
`[[:lower:]]`	Lower case
`[[:upper:]]`	Upper case
`[[:digit:]]`	Decimal digits
`[[:blank:]]`	Space / Tab
`[[:space:]]`	Whitespace
`[[:punct:]]`	Punctuation
`[[:xdigit:]]`	Hexadecimal
`[[:graph:]]`	Non-space
`[[:cntrl:]]`	Control characters

- Backreferences
  - `\1` through `\9`
  - `\n` matches whatever was matched by nth earlier paranthesized subexpression
  - A line with two occurances of hello will be matched using: `\(hello\).*\1`

BRE operator precedence

Highest to Lowest

[..] [==] [::] char collation

\metachar

[ ] Bracket expansion

( ) \n subexpresions and backreferences

* { } Repetition of preceding single char regex

Concatenation

^ $ anchors

ERE operator precedence

Highest to Lowest

[..] [==] [::] char collation

\metachar

[ ] Bracket expansion

( ) grouping

* + ? { } Repetition of preceding regex

Concatenation

^ $ anchors

| alternation

Examples using grep

- Basic use
  - `grep 'Raman' names.txt` matches line with Raman Singh
  - `cat names.txt | grep 'ai'` matches line with Snail
- Usage of `.`
  - `cat names.txt | grep 'S.n'` matches lines with Singh and Sankaran
- Usage of `$`
  - `cat names.txt | grep '.am$'` matches lines that end with xam
- Escaping a `.`
  - `cat names.txt | grep '\.'` matches lines that have a `.`
- Using anchors at the begining
  - `cat names.txt | grep '^M'` matches lines begining with m
- Case insensitive matching with the `i` flag
  - `cat names.txt | grep -i '^e'` matches lines begining with e or E.
- Word boundaries `\b`
  - `cat names.txt | grep 'am\b'` matches lines with words that end with 'am'

- Use of square brackets `[]` to give options
  - `cat names.txt | grep 'M[ME]'` matches lines containing 'MM' or 'ME'
  - `cat names.txt | grep '\bS.*[mn]'` matches lines containing words begining with S and ending with m or n.
  - `cat names.txt | grep '[aeiou][aeiou]'` matches lines that have 2 vowels side by side
  - `cat names.txt | grep 'B90[1-4]'` matches words begining with B90 and ending with range 1-4.
  - `cat names.txt | grep 'B90[^1-4]'` matches words begining with B90 and ending with characters other than the range 1-4. A hat inside square brackets implies negation
- Specifying occurances using escaped braces
  - `cat names.txt | grep 'M\{2\}'` matches lines which have 'MM'
  - `cat names.txt | grep 'M\{1,2\}'` matches lines which have one or 2 'M's
- Grouping patterns that are matched using parenthesis. Repeating whatever is matched by using `\1`
  - `cat names.txt | grep '\(ma\)'` matches lines containing 'ma'
  - `cat names.txt | grep '\(ma\).*\1'` matches a pattern begining with 'ma' and ending with 'ma' eg: U'mair Ahma'd. The `\1` back-references the first parenthesis.
  - `cat names.txt | grep '\(.a\).*\1'` matches a pattern like 'Mary Ma'nickam
  - `cat names.txt | grep '\(a.\)\{3\}'` matches a pattern like S'agayam'
- Using Extended Regular Expression Engine
  - `cat names.txt | egrep 'M+'` will match lines where M occures one or more times.
  - `cat names.txt | egrep '^M+'` will match lines where M occures one or more times at the begining of a line.
  - `cat names.txt | egrep '^M*'`
    - `cat names.txt | egrep '^M*a'` matches lines where 'M' may or may not occur followed by 'a'
    - `cat names.txt | egrep '^M.*a'` matches lines where 'M' has to occur at the begining of a line followed by any number of characters and ending with 'a'
    - Watch out for the interpretation of `*`
  - `cat names.txt | egrep '(ma)+'` 'ma' could occur one or more times.
  - `cat names.txt | egrep '(ma)*'` 'ma' could occur zero or more times.
- Use of pipe as an alternation between 2 patterns of strings to be matched
  - `cat names.txt | egrep '(ED|ME)'` matches lines containing 'ED' or 'ME'
  - `cat names.txt | egrep '(Anu|Raman)'` matches lines containing 'Anu' or 'Raman'. Length of string on both sides of pipe need not be the same.
  - `cat names.txt | egrep '(am|an)$'` matches lines containing 'am' or 'an' at the end.

4.4

More Examples using grep and egrep

- Get package names that are exactly 4 characters long

        - `dpkg-query -W -f'${Section} ${binary:Package}\n' | egrep ' .{4}$'`
    - Get package names that are from the math section
        - `dpkg-query -W -f'${Section} ${binary:Package}\n' | egrep '^math'`
    - get lines that have an alphanumeric character at the begining of the line
        - `cat chartype.txt | grep '^[[:alnum:]]'`
    - get lines that have digits at the end of the line
        - `cat chartype.txt | grep '[[:digit:]]$'`
    - get lines that have a ctrl character
        - `cat chartype.txt | grep '[[:ctrl:]]'`
        - `cat chartype.txt | grep -v '[[:ctrl:]]'` will show the reverse including the empty lines
    - get lines that do not have a ctrl character
        - `cat chartype.txt | grep '[^[:ctrl:]]'` (This does not work as intended)
    - get lines that have printable characters (exclude blank lines)
        - `cat chartype.txt | grep '[[:print:]]'`
    - get lines that have blank space characters (exclude blank lines)
        - `cat chartype.txt | grep '[[:blank:]]'`
    - `[[:graph:]]` is used to match any non space character
    - To skip blank lines
        - `cat chartypes.txt | egrep -v '^$'` Here `-v` excludes and `'^$'` captures empty lines
    - Identify a line with a 12 digit number
        - `egrep '[[:digit:]]{12}' patterns.txt`
    - Identify a line with a 6 digit number (Use word boundaries)
        - `egrep '\b[[:digit:]]{6}\b' patterns.txt`
    - Match lines containing Roll Number of the form MM22B001
        - `egrep '\b[[:alpha:]]{2}[[:digit:]]{2}[[:alpha:]][[:digit:]]{3}\b' patterns.txt`
    - Match urls without the http
        - `egrep '\b[[:alnum:]]+\.[[:alnum:]]+\b' patterns.txt`
    - **Trimming text**
        - top to bottom using `head` and `tail`
        - sidways or horizontal trimming of lines using `cut`
            - `cut -c 1-4 fields.txt` displays only first 4 characters. Can also use `-4` for begining to 4th place or `2-` to cut from 2nd place to end.
            - `cat fields.txt | cut -d " " -f 1` - This uses " " as a delimiter `-d` and prints only the first field `-f 1`
            - `cat fields.txt | cut -d ' ' -f 1-2` - to get both fields
            - Capture `hello world` from `1234;hello world,line 1`
                - `cat fields.txt | cut -d ';' -f 2 | cut -d "," -f 1`
                - `egrep ';.*,' fields.txt` (To trim pass the output of grep to `sed`)
            - Combining this with top to bottom trimming
                - `cat fields.txt | cut -d ';' -f 2 | cut -d "," -f 1 | head -n 2 | tail -n 1`

Own experiments using regex

- Get strictly alphanumeric words
  - `cat test.txt | egrep '\b([a-z]+[0-9]+|[0-9]+[a-z]+)\b'`