

# CS 35L

LAB 8, Session 2

TA: Sucharitha Prabhakar

EMAIL ID: [prabhakarsucharitha@gmail.com](mailto:prabhakarsucharitha@gmail.com)

# Outline

- Unix wildcards, basic regular expressions
- Commands - Grep, sed
- Lab assignment



# Details

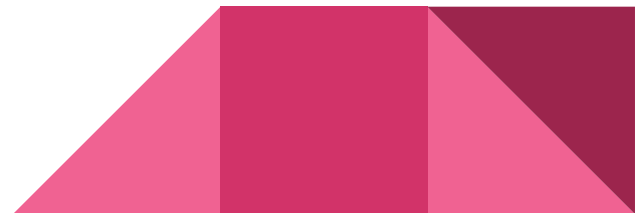
`mkdir lab2_2` (Make a directory for each lab session)

`cd lab2_2`

`touch lab.log` (optional)

`touch lab.txt`

`touch hw.txt`



# UNIX WILDCARDS

# Wildcard characters

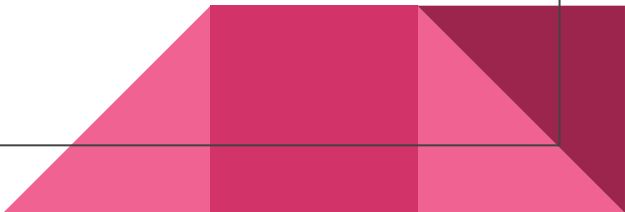
Commands can use wildcards to perform actions on more than one file at a time, or to find part of a phrase in a text file

**Standard Wildcard characters** - used by various command-line utilities to work with multiple files.

man 7 glob

**Regular expressions** - used to manipulate parts of text.

man 7 regex



# Standard Wildcard

- ? (question mark) - Represents any *single* character.
  - Example: "ab?" - Look for aba, abb, abc and ab followed by every other letter/number between a-z, 0-9.
- \* (asterisk) - Represent any number of characters (including zero, in other words, zero or more characters).
  - Example - "ab\*" it would use "aba", "abcde", "abczyta" and *anything* that starts with "ab" also including "ab" itself.
  - "p\*l" could be pill, pull, pl, and anything that starts with an p and ends with an l.
- [] (square brackets) - Specifies a range.
  - Example - If you did r[a,o,i]m it can become: ram, rim, rom
  - If you did: r[a-d]m it can become anything that starts and ends with m and has any character a to d in between.

# Standard Wildcard

- `{ }` (curly brackets) - Terms are separated by commas and each term must be the name of something or a wildcard.
  - Example - `{*.doc,*.pdf}` - Anything ending with `.doc` or `.pdf`. Note that spaces are not allowed after the commas (or anywhere else). `cp {*.doc,*.pdf} ~` (This wildcard will copy anything that matches either wildcard(s), or exact name(s) (an “or” relationship, one or the other).
- `[!]` - This construct is similar to the `[ ]` construct, except rather than matching any characters inside the brackets, it'll match any character, as long as it is not listed between the `[` and `]`. This is a logical NOT.
  - Example `rm myfiles[!9]` will remove all `myfiles*` (ie. `myfiles1`, `myfiles2` etc) but won't remove a file with the number 9 anywhere within it's name.

# Standard Wildcard

- \ (backslash) - Used as an "escape" character, i.e. to protect a subsequent - special character.
  - Example - Thus, "\\" searches for a backslash.





# Regular expressions

- . (dot) - Will match *any single character*, equivalent to ? (question mark) in standard wildcard expressions.
  - Example - "a.c" matches "aac" and "abc" but not "ac" or "abbc".
- \ (backslash) - Used as an "escape" character, i.e. to protect a subsequent special character. Thus, "\\" searches for a backslash.
- \* (asterisk) - The preceding item is to be matched *zero or more* times. ie. n\* will match n, nn, nnnn, nnnnnnnn but not na or any other character.
- .\* (dot and asterisk) - Used to match any string, equivalent to \* in standard wildcards.
- ^ (caret) - The beginning of the line".
  - Example - "^a" means find a line starting with an "a".

# Regular expressions

- **\$ (dollar sign)** - The end of the line".
  - Example - "a\$" means a line ending with an "a".
- **[] (square brackets)** - Specifies a range.
  - Example - If you did `m[a,o,u]m` it can become: mam, mum, mom if you did: `m[a-d]m` it can become anything that starts and ends with m and has any character a to d in between. This kind of wildcard specifies an "or" relationship (you only need one to match).
- **|** - This wildcard makes a logical OR relationship between wildcards. You may need to add a `\` (backslash) before this command to work, because the shell may attempt to interpret this as a pipe.
- **[^]** - Equivalent of `[!]` in standard wildcards. This performs a logical "not". This will match anything that is not listed within those square brackets.
  - Example - `rm myfiles[^9]` will remove all `myfiles*` (ie. `myfiles1`, `myfiles2` etc) but won't remove a file with the number 9 anywhere within it's name.

# Examples

Expression	Matches
<code>tolstoy</code>	The seven letters <code>tolstoy</code> , anywhere on a line
<code>^tolstoy</code>	The seven letters <code>tolstoy</code> , at the beginning of a line
<code>tolstoy\$</code>	The seven letters <code>tolstoy</code> , at the end of a line
<code>^tolstoy\$</code>	A line containing exactly the seven letters <code>tolstoy</code> , and nothing else
<code>[Tt]olstoy</code>	Either the seven letters <code>Tolstoy</code> , or the seven letters <code>tolstoy</code> , anywhere on a line
<code>tol.toy</code>	The three letters <code>tol</code> , any character, and the three letters <code>toy</code> , anywhere on a line
<code>tol.*toy</code>	The three letters <code>tol</code> , any sequence of zero or more characters, and the three letters <code>toy</code> , anywhere on a line (e.g., <code>toltoy</code> , <code>tolstov</code> , <code>tolWHOtoy</code> , and so on)

# Character Categories

- [:upper:] uppercase letters
- [:lower:] lowercase letters
- [:alpha:] alphabetic (letters) meaning upper+lower (both uppercase and lowercase letters)
- [:digit:] numbers in decimal, 0 to 9
- [:alnum:] alphanumeric meaning alpha+digits (any uppercase or lowercase letters or any decimal digits)
- [:space:] whitespace meaning spaces, tabs, newlines and similar




# Character Categories

- `[:punct:]` punctuation characters meaning graphical characters minus alpha and digits
- `[:cntrl:]` control characters meaning non-printable characters
- `[:xdigit:]` characters that are hexadecimal digits.

Example: `ls -l | grep '[:upper:][:digit:]'`

The command greps for `[upper_case_letter][any_digit]`, meaning any uppercase letter followed by any digit



# Commands

# Grep

Search a file for a pattern

Some useful options:

- -F : Match using fixed strings.
- -e pattern\_list : Specify one or more patterns to be used during the search for input.
- -f pattern\_file : Search from a file
- -i : Perform pattern matching in searches without regard to case



# Grep

- **-n** Precede each output line by its relative line number in the file, each file starting at line 1.
- **-q** No output to stdout
- **-v** Select lines not matching any of the specified patterns.
- **-x** Consider only input lines that use all characters to match an entire fixed string or regular expression to be matching lines.

The standard input shall be used if no *file* operands are specified



# Grep

```
grep -E '^Bat' myfile.txt
```

```
grep -E 'man$' myfile.txt
```

```
grep -E '^(bat|Bat|cat|Cat)' myfile.txt
```

```
grep -i -E '^(bat|cat)' myfile.txt
```

```
grep -E '^[bcBC]at' myfile.txt
```

```
grep -i -E '^[^b]at' myfile.txt
```



# Sed

**Sed** is a stream editor. Used to perform basic text transformations on an input stream (a file or input from a pipeline).

```
sed 's/regExpr/replText/[g]'
```



# Sed Examples

- `sed 's/Nick/John/g' report.txt` - Replace every occurrence of Nick with John in report.txt
- `sed 's/[N|n]ick/John/g' report.txt` - Replace every occurrence of Nick or nick with John.
- `sed '$d' file.txt` - Delete the last line
- `sed '/[0-9]\{3\}/p' file.txt` - Print only lines with three consecutive digits
- `sed '17,/disk/d' file.txt` - Delete all lines from line 17 to 'disk'



# Other important commands

wc: outputs a one-line report of lines, words, and bytes

head: extract top of files

tail: extracts bottom of files

tr: translate or delete characters

sort: sort lines of text files

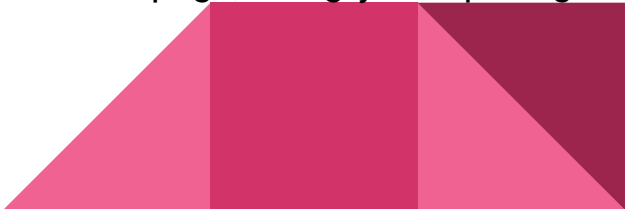


# LAB 2

# Introduction

Build a spelling checker for the Hawaiian language  
(Get familiar with sort, comm and tr commands!)

Steps:

- Download a copy of web page containing basic English-to-Hawaiian dictionary
  - Extract only the Hawaiian words from the web page to build a simple Hawaiian dictionary. Save it to a file called hwords (site scraping)
  - Automate site scraping: buildwords script (`cat file.html | buildwords > hwords`)
  - Modify the command in the lab assignment to act as a spelling checker for Hawaiian
  - Use your spelling checker to check hwords and the lab web page for spelling mistakes
  - Count the number of "misspelled" English and Hawaiian words on this web page, using your spelling checkers.
- 

# Hints

- Run your script on seasnet servers before submitting to CCLE
- `sed '/patternstart/,/patternstop/d'`
  - delete patternstart to patternstop, works across multiple lines. Will delete all lines starting with patternstart to patternstop
- The Hawaiian words html page uses `\r` and `\n` for new lines
  - `od -c hwnwdseng.htm` to see the ASCII characters
- You can delete blank white spaces such as tab or space using
  - `tr -d '[:blank:]'`
- Use `tr -s` to squeeze multiple new lines into one
- `sed 's/<[^>]*>//g' a.html` to remove all HTML tags

# References

<http://www.tldp.org/LDP/GNU-Linux-Tools-Summary/html/x11655.htm>

<https://www.ibm.com/developerworks/aix/library/au-speakingunix9/>

<https://linuxconfig.org/learning-linux-commands-sed>

