EECS 2031

Software Tools

Module 1 – Intro to Unix

YORK
UNIVERSITÉ
UNIVERSITY

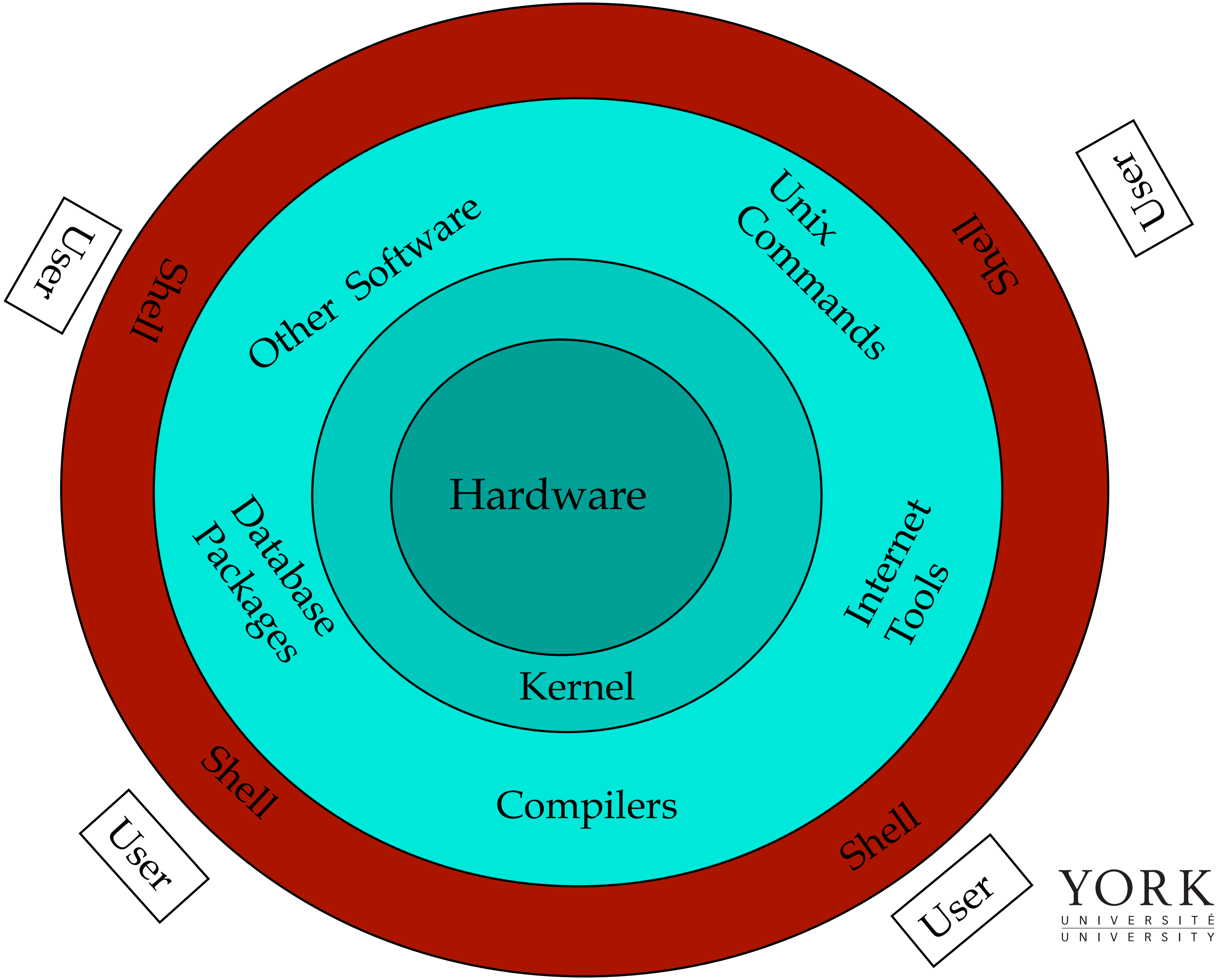
What is UNIX?

- An Operating System (OS)
- Mostly coded in C
- It provides a number of facilities:
  - Management of hardware resources
  - Directory and file system
  - Execution of programs

GUI vs. Command Line

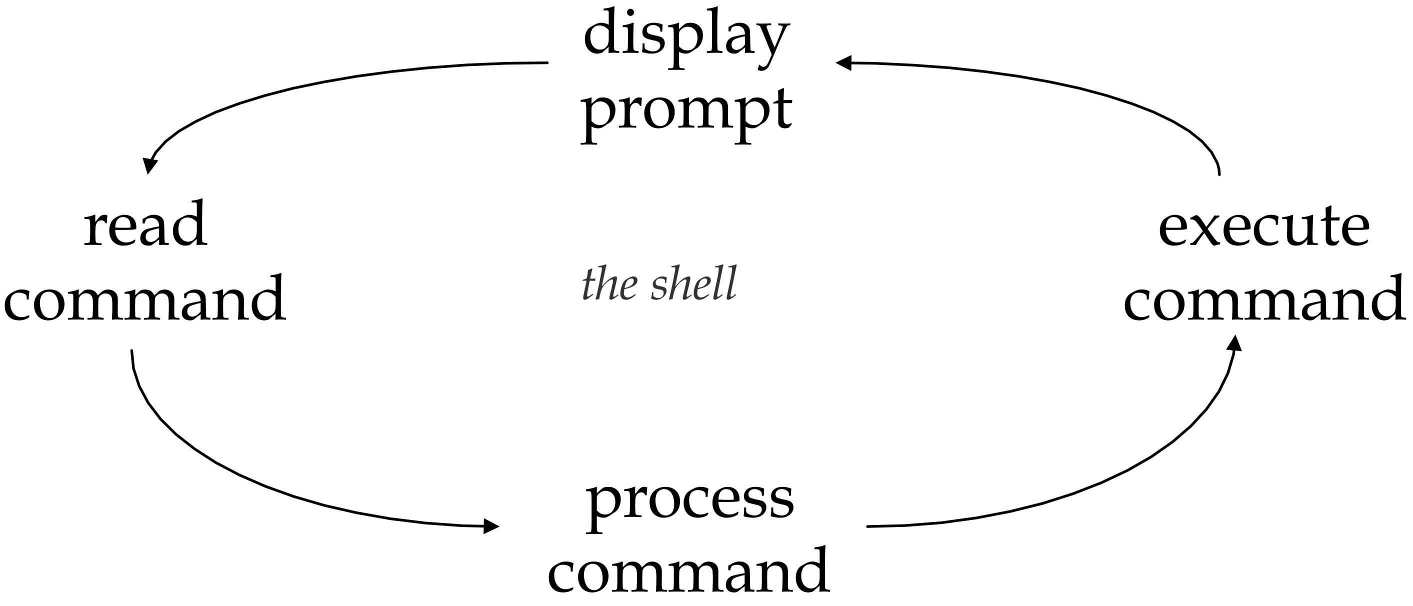
- When you log in to your EECS account, you get a graphical interface built on top of the Unix ***kernel***
- In this course, we are concerned with the command line interface of Unix
- You access it with the help of the ***shell***, a program that runs every time you open a Terminal window

Kernel-Shell Relationship



The Shell

- The shell does 4 jobs repeatedly:



Unix Commands

- There are many of them
- We will see some of the most useful ones
- The very basics:
- `ls, cp, mv, rm`
- `cd, pwd, mkdir, rmdir`
- `man`

Some more commands

• `date`	Gives time and date

• `cal`
`cal 1969`
`cal 7 2011`
	Calendar

• `passwd`	Changes your password

You and the System

- `uptime` Machine’s ‘up’ time
- `hostname` Name of the machine
- `whoami` Your name
- `who`

history

```
% history 8
   325   12:48    ls
   326   12:48    m ex1.c
   327   12:49    who
   328   12:50    history 10
   329   12:52    ls -a
   330   12:56    ls Stack/
   331   12:57    ls
   332   12:57    history 8
```

echo

- When one or more strings are provided as arguments, echo by default repeats those strings on the screen.

```
% echo This is a test.

This is a test.
```

- It is not necessary to surround the strings with quotes, as it does not affect what is written on the screen.

- If quotes (either single or double) are used, they are not repeated on the screen.

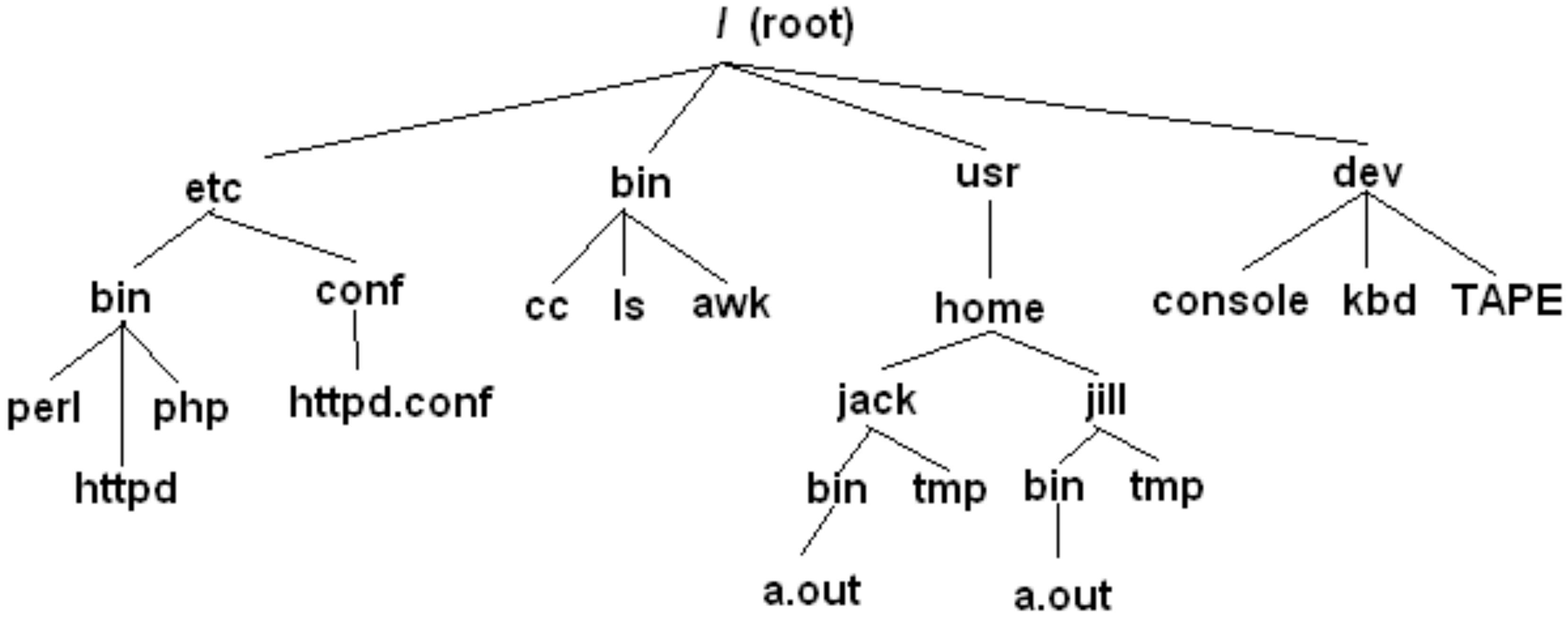
```
% echo 'This is' "a test. "

This is a test.
```

- To display single/double quotes, use `\'` or `\"`

The File System

- Directory structure
- Current working directory
- Path names
- Special notations

Directory Structure



Current Working Directory

- In a shell, the command **ls** shows the contents of the current working directory.
- **pwd** shows the full path of the current working directory.
- **cd** changes the current working directory to another.

Path Names

- A path name is a reference to something in the file system.
- A path name specifies the set of directories you have to pass through to find a file.
- Directory names are separated by / in UNIX.
- Path names beginning with / are **absolute** path names.
- Path names that do not begin with / are **relative** path names (start search in current working directory).

Special Characters

- . refers to the current directory
- .. refers to the parent directory
  - **cd ..**
  - **cd ../Notes**
- ~ refers to the home directory
  - **cat ~/lab3.c**
- To go directly to your home directory, type
  - **cd**

Wildcards (File Name Substitution)

- Allow user to refer to several files at once
- How to list all files in the current directory that start with ‘a’?

```
ls a*
```

? – Matches single character

- `ls a?.txt`

```
a1.txt  a2.txt ab.txt
```

- `ls lab1.???`

```
lab1.doc lab1.pdf
```

* - Matches several characters

-
  ```
  ls a*.txt
  ```

  ```
  a1.txt  a2.txt  abcd.txt
  abc.txt  a.b.txt  ab.txt
  ```

-
  ```
  ls lab1.*
  ```

  ```
  lab1. lab1.c lab1.doc
  lab1.docx lab1.pdf
  ```

[…] – Matches all listed characters

- `ls lab[123].pdf`

```
lab1.pdf   lab2.pdf   lab3.pdf
```

- `ls a[ab]*.???`

```
abcd.txt   abc.txt   ab.txt
```

cat, more

```
% cat phone_book
Yvonne 416-987-6543
Amy 416-123-4567
William 905-888-1234
John 647-999-4321
Annie 905-555-9876
```

```
% more phone_book
```

Similar to cat, except that the file is displayed one screen at a time.

tail, head

```
% tail phone_book
```

Display the last 10 lines

```
% tail -5 phone_book
```

Display the last 5 lines

```
% tail -1 phone_book
```

Display the last line

```
% tail –n +13 phone_book
```

Display the file starting from the 13th line.

head is similar for the beginning of the file

wc

```
% wc a1.txt
 12 13 68 a1.txt


% wc *.pdf
 12   13   68 lab1.pdf
 17   18  101 lab2.pdf
 17   31  165 lab3.pdf
 46   62  334 total
```

```
% wc -c a1.txt
68 a1.txt


% wc -w a1.txt
13 a1.txt


% wc -l a1.txt
12 a1.txt
```

cmp, diff

```
% cmp file1 file2
file1 file2 differ: char 9, line 2
% diff phone_book phone_book2
2c2
< Amy 416-123-4567
---
> Amy 416-111-1111
```

Stdin / Stdout

- Each Unix command reads input from standard input (stdin) and produces output to standard output (stdout)
- By default, stdin is the keyboard, and stdout is the screen
- But this can change…

Input / Output Redirection

- Redirect output to a file (overwriting)
  - `ls > all_files.txt`
- Append output to a file
  - `ls >> all_files.txt`
- Read input from a file
  - `wc < all_files.txt`

Pipes

- A way to connect the output of one program to the input of another program without a temporary file.

`ls -1 | wc -l`	count number of files
`who | sort`	sort user list
`who | wc -l`	count users

Command Terminators

- New line or ; - Execute in order

```
% date; who
```

- & - Do not wait for command to complete

```
% nedit lab9.c&
```

  - Used to put a long-running command “in the background” while you continue to use the terminal for other commands.

Single Quotes

- What's the difference between these two commands?

```
% ls  a*t

% ls 'a*t'
```

- Quotes do not have to surround the whole argument.

```
% echo a'*'t

a*t
```

Double Quotes

- Double quotes can also be used to protect special characters, but …
- The shell will interpret `$`, `\` and \`…\` inside the double quotes.
- So don’t use double quotes unless you intend some processing of the quoted string.

sort

```
% cat phone_book
Yvonne 416-987-6543
Amy 416-123-4567
William 905-888-1234
John 647-999-4321
Annie 905-555-9876
```

```
% sort phone_book
Amy 416-123-4567
Annie 905-555-9876
John 647-999-4321
William 905-888-1234
Yvonne 416-987-6543
```

`sort` – Useful options

`sort -r`	reverse normal order
`sort -n`	numeric order
`sort -nr`	reverse numeric order
`sort -f`	case insensitive

uniq

- Removes repeated lines in a file

```
uniq [-c] [input [output]]
```

- Notice difference in args:
  - 1st filename is input file
  - 2nd filename is output file
- If input is not specified, use stdin
- If output is not specified, use stdout

`uniq`

- Only works for lines that are adjacent, e.g.

```
abacus

abacus

bottle

abacus
```

becomes

```
abacus

bottle

abacus
```

`uniq`

- With the `-c` option output is a count of how many times each line was repeated
- For previous input:

```
2 abacus
1 bottle
1 abacus
```

`sort` + `uniq`

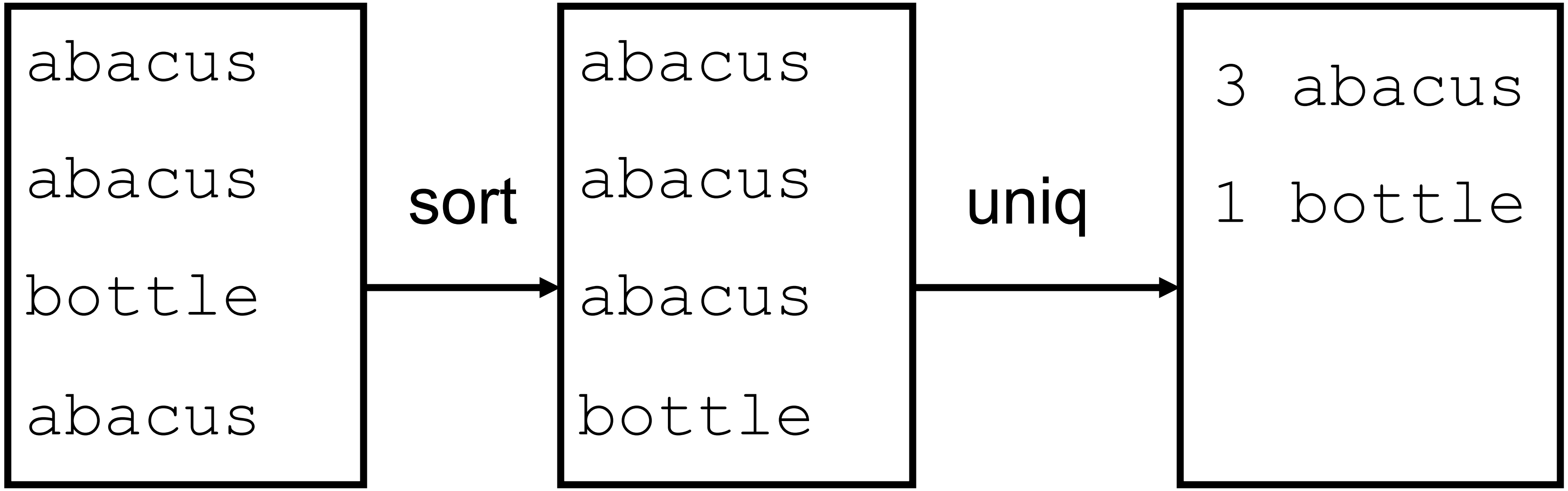
- `uniq` is a little limited but we can combine it with `sort`

```
sort | uniq -c
```

- Counts number of times line appears in file
- Output would now be:

```
3 abacus

1 bottle
```

sort + uniq



```
abacus
abacus
bottle
abacus
```

sort

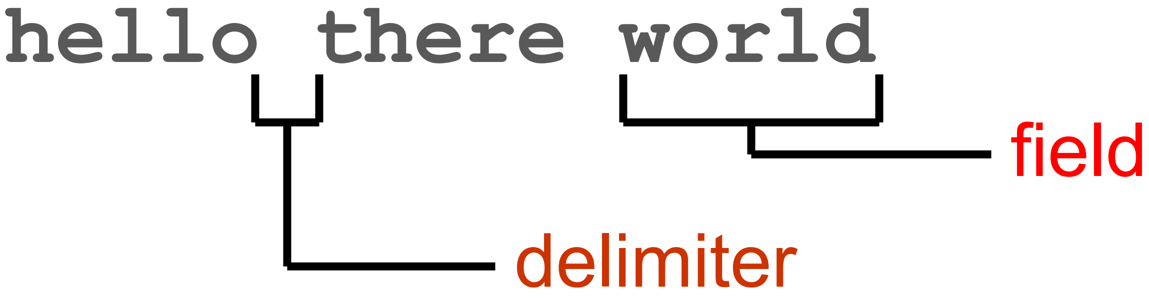
```
abacus
abacus
abacus
bottle
```

uniq

```
3 abacus
1 bottle
```

cut

- Used to split lines of a file
- A line is split into fields
- Fields are separated by delimiters
- A common case where a delimiter is a space:



```
hello there world
```

field

delimiter

`cut` - Syntax

- **`cut [-f`***`fields`***`] [-c`***`columns`***`]`**
  `[-d`*`delimiter`***`] [`***`filename`*** **`…]`**
- If filenames are given on command line, input is taken from those files
- If no filenames are given, input comes from stdin
- This approach to input is very common

`cut` – Extracting fields

```
cut -f3 -d,
```

- Extract field 3 from each line
- Fields are separated by commas
- With an input of

```
hello,there,world,!
```

- output would be just **`world`**

cut – Extracting characters

```
cut -c30-40
```

- Extract characters 30 through 40 (inclusive) from each line
- Note that we can use ranges (e.g. 4-10) or lists (e.g. 4,6,7) as values for **`-f`** or **`-c`**.

tr

- Maps characters from one value to another

```
tr string1 string2
```

```
tr [-d] [-c] string
```

- Input is always stdin, output is always stdout
- A character in string1 is changed to the corresponding character in string2

tr

- A simple example:

```
    tr x y
```

- All instances of **x** are replaced with **y**

- Each string can be a set of characters

```
    tr ab xy
```

- **a** is replaced with **x**, **b** is replaced with **y**

`tr`

- The `-d` option means delete the given characters

```
    tr -d xyz
```

- Delete all **`x`**, **`y`**, and **`z`** characters
- The `-c` option means "complement”

```
    tr -d -c xyz
```

- Delete everything except **`x`**, **`y`**, and **`z`**

Why Are These So Weird?

- Unix philosophy:

  Do one thing and do it well

- **`tr`** doesn't know how to read from files, but the **`cat`** command does:

```
cat filename | tr …
```

grep

- Outputs all lines in the input that match the given *regular expression*

  grep [*options*] *regex* [*file* …]

  e.g.

```
grep hello *.txt
```

outputs all lines containing **hello** in any file that ends in **.txt** in the current directory

Regular Expressions

- A regular expression is a special string (a sequence of characters)
- Describes a search pattern, i.e. each regular expression matches a set of strings
- **`grep`** uses regular expressions to search the contents of files
- Looks like wildcards but is ***quite different***!

Literals

- Letters and numbers are literal - that is they match themselves:
- The regular expression

  `foobar`

  matches only the string

  `foobar`

. – Matches exactly one character

- The regular expression

  ```
  fooba.
  ```

  Matches the following strings

  ```
  foobar
  foobat
  foobay
  ```

  etc.

. – Matches exactly one character

- Each dot must match exactly one character
- The regular expression

  `f..bar`

  matches

  `foobar` or **`fWRbar`**

  but not

  `fubar` or **`fooobar`**

[ ] – Matches any listed character

- The regular expression

  ```
  foob[aeiou]r
  ```

  matches only the 5 strings

  ```
  foobar
  foober
  foobir
  foobor
  foobur
  ```

* - 0 or more of the last character

- The regular expression

  ```
  fo*
  ```

  matches

  ```
  f
  fo
  foo
  fooo
  foooo
  ```

  etc.

* - 0 or more of the last character

- The regular expression
  ```
  [0-9][0-9]*
  ```
  matches all decimal numbers including ones with leading zeros such as
  ```
  000042
  ```

* - 0 or more of the last character

- The regular expression

  ```
  .*
  ```

  matches anything
  including an empty string

^ $ - Beginning and end of line

- The regular expression

  ```
  ^foobar
  ```

  matches any line that *starts* with

  ```
  foobar
  ```

- The regular expression

  ```
  foobar$
  ```

  matches any line that *ends* with

  ```
  foobar
  ```

grep

- Let's say you want to search for any string that starts with b followed by 0 or more a's in file a.txt
- The following will not work

```
grep ba* a.txt
```

- Why not?

`grep` Options

- `-i` case-insensitive search (don't distinguish between **`a`** and **`A`**)
- `-v` invert search (output lines which don't match)
- `-l` Output only the names of files with matching lines
- `-c` Output only the number of lines that match

grep – Interesting Uses

```
    grep -v '^#'
```

Removes all lines beginning with '#'

```
    grep -v '^[ ]*$'
```

Removes all lines which are either empty or contain only spaces

fgrep (faster grep)

- Like grep, fgrep searches for things but does not do regular expressions - just fixed strings

```
fgrep 'hello.*goodbye'
```

Searches for string “hello.*goodbye” - does not match it as a regular expression

egrep (extended grep)

- **grep** interprets only basic regular expressions.
- Extended regular expressions use additional metacharacters to allow expression of more elaborate search patterns
- Use **egrep** if you require this

? – 0 or 1 of the last character

- The regular expression

  ```
  [1-9][0-9]?
  ```

  matches all numbers from `1` to `99`

- The regular expression

  ```
  colou?r
  ```

  matches

  ```
  color
  colour
  ```

| - Used as an OR

- The extended regular expression

  ```
  0|[1-9][0-9]?
  ```

  matches all numbers from `0` to `99`

- Parentheses can be used as well

Finding Files

- Wildcards are limited
- The following commands help us to find files and run commands on them

`find`

- Finds files with given properties

```
find path [expression …]
```

- Not just regular files - includes directories, devices - everything it finds in the filesystem
- Starts at the given path and examines every file in every directory it finds recursively

`find`

- We can specify expressions to designate which files we are interested in and what to do with them
- All expressions begin with a dash

```
find ~ -print
```

Outputs the name of every file in your home directory (including subdirectories)

`find`

- Expressions are handled left-to-right
- For each file examined, each expression is evaluated as true or false
- Stop processing for a file if an expression is false
- e.g. **`-empty`** evaluates to true if the file is empty, false otherwise

`find`

- Another expression: **`-type`** *`filetype`*
- True if the examined file is of the specified type
- **`f`** = regular file, **`d`** = directory

```
    find ~ -type d -empty
```

Outputs all empty directories under your home directory.

`find`

`-name` *`pattern`*
true if the name of the file matches the wildcard pattern

```
find ~ -type f -name '*.doc'
```

Finds all files under your home directory which are regular files and end in **`.doc`**

xargs

- Syntax: **`xargs`** *`command`*
- Executes given command for each word in its stdin

```
find ~ -type f -name '*.txt’ |
xargs wc -l
```

Counts number of lines in all **`.txt`** files

File permissions

- Try `ls -l`
- Each file comes with a 10-character string

```
-rwxr--r--
```

The owner of this file can read, write, and execute this file, but everybody else can only read it

File/Directory Permissions

Letter	Meaning
r	Permission to read the file or the contents of a directory
w	Permission to write to the file, or create a new file in a directory
x	For a file: permission to execute For a directory: permission to enter the directory and execute programs in it

Changing Permissions

Letter	Meaning
u	The user who owns the file
g	The group the file belongs to
o	The other users
a	All of the above

chmod Command

```
chmod who+permissions filename

chmod who-permissions filename


chmod u+x my_script

chmod a+r index.html

chmod a+rx Notes/
```

`chmod` with Binary Numbers

```
chmod UGO myFile
```

U = a number from 0 to 7 whose binary representation denotes the read, write, and execute permissions for the user

G,O = Same for group and others

```
chmod 644 myFile
```

6 means the user can read and write

Group and others can only read

chgrp Command

A file owner can change the group a file belongs to

```
chgrp grp_name filename
```

Examples:

```
chgrp submit lab1
```

```
chgrp labtest lab9
```

`id` Command

To display the group(s) a user belongs to, use the `id` command:

```
% id cse12345
```

```
uid=12695(cse12345)
gid=10000(ugrad)
groups=10000(ugrad)
```

Processes

- Each running program on a UNIX system is called a process.
- Processes are identified by a number (process id or PID).
- Each process has a unique PID.
- There are usually several processes running **concurrently** in a UNIX system.

ps command

```
% ps a
```

list all processes

```
  PID TTY      STAT   TIME COMMAND
 2763 pts/11    S+    0:10 pine
14468 pts/19    R+    0:00 ps
14780 pts/21    S     0:00 xterm
26772 pts/2     S+    0:01 emacs
  ...
```

Background processes

- A process may be in the foreground, in the background, or be suspended
- To see all background processes: **`jobs`**
- To bring a process to the foreground: **`fg`**
- To suspend the foreground process: **`CTRL-Z`**
- Put all suspended processes to the background: **`bg`**

kill

```
% kill -KILL PID
```

to terminate a process

```
% kill -STOP PID
```

to suspend a process

Frequently Used Terminal Keystrokes

- Kill the current process: **`CTRL-C`**
- Suspend the current process: **`CTRL-Z`**
- End of input: **`CTRL-D`**

Homework

- Activate your EECS account before the lab (instructions on course webpage)
- Login to your EECS account and try all these commands
- Read the tutorials posted as part of the labs
- Answer lab questions