Unix
Bash
C
GNU
Systems

COMP 206 – Joseph Vybihal
Software Systems

Software Systems

Lectures Week 2

Regular Expressions, Developer Techniques and Intro to Bash Scripting

Prof. Joseph Vybihal
Computer Science
McGill University

Part 1

Regular Expressions & Wild Cards

Readings: http://www.thegeekstuff.com/2011/01/regular-expressions-in-grep-command/

Unix
Bash
C
GNU
Systems

COMP 206 – Joseph Vybihal
Software Systems

Wild Cards

The ability to select multiple files with a single expression.

Unix
Bash
C
GNU
Systems

COMP 206 – Joseph Vybihal
Software Systems

Wild cards

- The “asterix”
- ls *.doc
- The “question mark”
- ls *.d?c
- The “square brackets”
- ls *.d?[abc]

* any pattern
? any single char
[ ] or

Wild cards

- The “asterix”
- ls *.doc
- The “question mark”
- ls *.d?c
- The “square brackets”
- ls *.d?[abc]

* any pattern
? any single char
[ ] or

John.doc
Bill.dla
Mary.dzc

Let us try it on the command-line

```
ls
cp
```

Incorporate paths

Regular Expressions

Like wild cards but more advanced.

It can be used with file names (like wild cards), but more importantly it can be used in searching, string manipulation, and text file manipulation.

Unix
Bash
C
GNU
Systems

COMP 206 – Joseph Vybihal
Software Systems

Regular Expressions

- Several Unix commands and editors allow you to search on text patterns.
- These text patterns are known as regular expressions (or *regex*).

These are popular commands that use regular expressions

- grep [options] STRING   FILE_LIST
  - search for occurrences of the string (we will only study this one)
- sed [options] FILE_LIST
  - stream editor for editing files.
- awk [options] FILE_LIST
  - scan for patterns in a file and process the results (script execution)

Unix
Bash
C
GNU
Systems

COMP 206 – Joseph Vybihal
Software Systems

Grep

- `grep` is used to search for the patterns in files.

- Regular expressions, are best specified in apostrophes (or single quotes) when used with grep.

- Some common options include:
  - -i : ignore case
  - -c : report only a count of the number of lines containing matches
  - -v : invert the search, displaying only lines that do not match
  - -n : display the line number along with the line on which a match was found
  - -l : list filenames, but not lines, in which matches were found

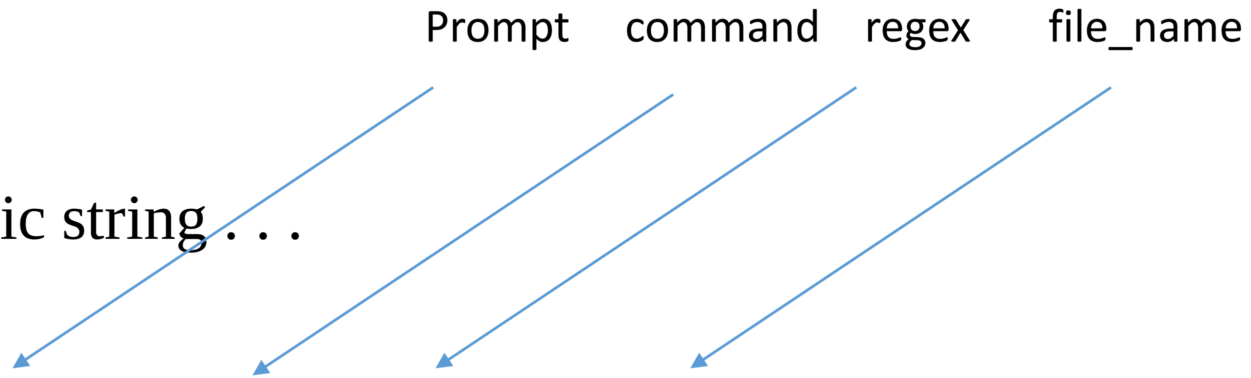
Example using grep

- Consider the following text file :

```
Alex
Marc
Micheal
Ting
Juan
Jeremy
Jessica
Yannick
Nicolas
Jean-Sebastien
Nadeem
```

Examples of grep (cont.)

•Grep for a specific string . . .



```
[jvybihal][~/cs206] grep ’Je’ demo.txt
Jeremy
Jessica
Jean-Sebastien
```

Notice quotation around the regular expression.

Unix
Bash
C
GNU
Systems

COMP 206 – Joseph Vybihal
Software Systems

Examples of grep (cont.)

- Grep for a specific string . . .

```
[jvybihal][~/cs206] grep ’Je’ demo.txt
Jeremy
Jessica
Jean-Sebastien

[jvybihal][~/cs206] grep -n ’Je’ demo.txt
6:Jeremy
7:Jessica
10:Jean-Sebastien

[jvybihal][~/cs206] grep -c ’Je’ demo.txt
3
```

Examples of grep (cont.)

- Grep for vowels . . .

```
[jvybihal][~/cs206] grep -i ’^[aeiouy]’ demo.txt
Alex
Yannick

[jvybihal][~/cs206] grep -i ’[aeiouy]$’ demo.txt
Jeremy
Jessica

[jvybihal][~/cs206] grep -i ’[aeiouy]{2,}’ demo.txt
Micheal
Juan                         '*[aeiouy]*[aeiouy]*'
Yannick
Jean-Sebastien
Nadeem
```

Examples of grep (cont.)

- Grep for specific characters . . .

```
[jvybihal][~/cs206] grep -i ’^.e’ demo.txt
Jeremy
Jessica
Jean-Sebastien

[jvybihal][~/cs206] grep -i ’^.e|a.$’ demo.txt
Micheal
Juan                    '^.[a-e]|a.$'
Jeremy
Jessica
Nicolas
Jean-Sebastien
```

Literal Characters

\f	Form feed
\n	Newline (Use \p in UltraEdit for platform independent line end)
\r	Carriage return
\t	Tab
\v	Vertical tab
\a	Alarm (beep)
\e	Escape
\xxx	The ASCII character specified by the octal number xxx
\xnn	The ASCII character specified by the hexadecimal number nn
\cX	The control character ^X. For example, \cI is equivalent to \t and \cJ is equivalent to \n

Character Classes

[...]	Any one character between the brackets.
[^...]	Any one character not between the brackets.
.	Any character except newline. Equivalent to [^\n]
\w	Any word character. Equivalent to [a-zA-Z0-9_] and [[:alnum:]_]
\W	Any non-word character. Equivalent to [^a-zA-Z0-9_] and [^[:alnum:]_]
\s	Any whitespace character. Equivalent to [\t\n\r\f\v] and [[:space:]]
\S	Any non-whitespace. Equivalent to [^ \t\n\r\f\v] and [^[:space:]] Note: \w != \S
\d	Any digit. Equivalent to [0-9] and [[:digit:]]
\D	Any character other than a digit. Equivalent to [^0-9] and [^[:digit:]]
[\b]	A literal backspace (special case)
[[:class:]]	alnum alpha ascii blank cntrl digit graph lower print punct space upper xdigit

Replacement

\	Turn off the special meaning of the following character.
\n	Restore the text matched by the nth pattern previously saved by \(and \). n is a number from 1 to 9, with 1 starting on the left.
&	Reuse the text matched by the search pattern as part of the replacement pattern.
~	Reuse the previous replacement pattern in the current replacement pattern. Must be the only character in the replacement pattern. (ex and vi).
%	Reuse the previous replacement pattern in the current replacement pattern. Must be the only character in the replacement pattern. (ed).
\u	Convert first character of replacement pattern to uppercase.
\U	Convert entire replacement pattern to uppercase.
\l	Convert first character of replacement pattern to lowercase.
\L	Convert entire replacement pattern to lowercase.

Repetition

{n,m}	Match the previous item at least n times but no more than m times.
{n,}	Match the previous item n or more times.
{n}	Match exactly n occurrences of the previous item.
?	Match zero or one occurrences of the previous item. Equivalent to {0,1}
+	Match one or more occurrences of the previous item. Equivalent to {1,}
*	Match zero or more occurrences of the previous item. Equivalent to {0,}
{}?	Non-greedy match - will not include the next match's characters.
??	Non-greedy match.
+?	Non-greedy match.
*?	Non-greedy match. E.g. ^(.*?)\s*$ the grouped expression will not include trailing spaces.

Options

g	Perform a global match. That is, find all matches rather than stopping after the first match.
i	Do case-insensitive pattern matching.
m	Treat string as multiple lines (^ and $ match internal \n).
s	Treat string as single line (^ and $ ignore \n, but . matches \n).
x	Extend your pattern's legibility with whitespace and comments.

Extended Regular Expression

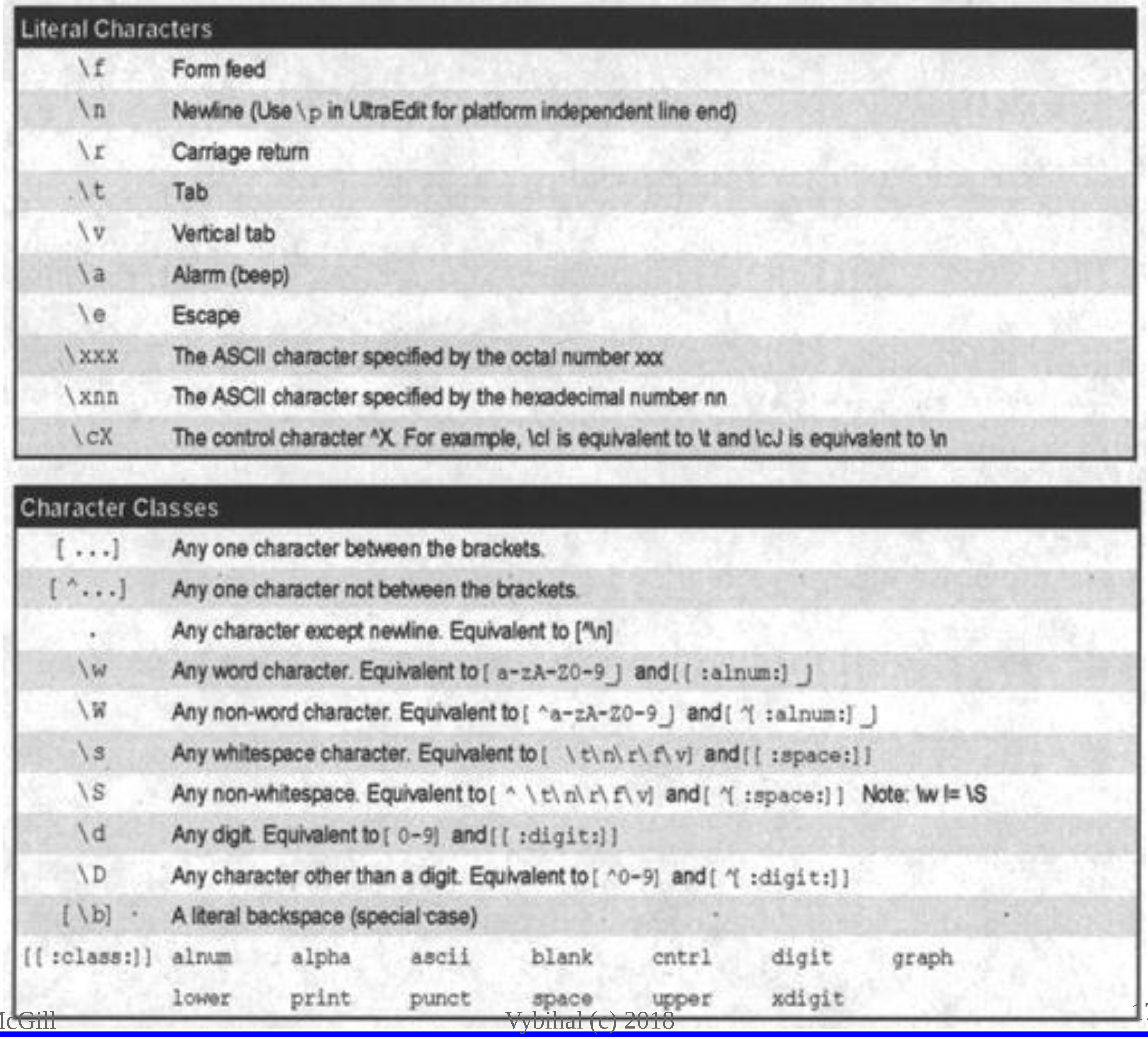
(?#...)	Comment, "..." is ignored.
(?:...)	Matches but doesn't return "..."
(?=...)	Matches if expression would match "..." next
(?!...)	Matches if expression wouldn't match "..." next
(?imsx)	Change matching rules (see options) midway through an expression.

Grouping

(...)	Grouping. Group several items into a single unit that can be used with *, +, ?, |, and so on, and remember the characters that match this group for use with later references.
|	Alternation. Match either the subexpressions to the left or the subexpression to the right.
\n	Match the same characters that were matched when group number n was first matched. Groups are subexpressions within (possibly nested) parentheses.

Anchors

^	Match the beginning of the string, and, in multiline searches, the beginning of a line.
$	Match the end of the string, and, in multiline searches, the end of a line.
\b	Match a word boundary. That is, match the position between a \w character and a \W character. (Note, however, that [\b] matches backspace.)
\B	Match a position that is not a word boundary.



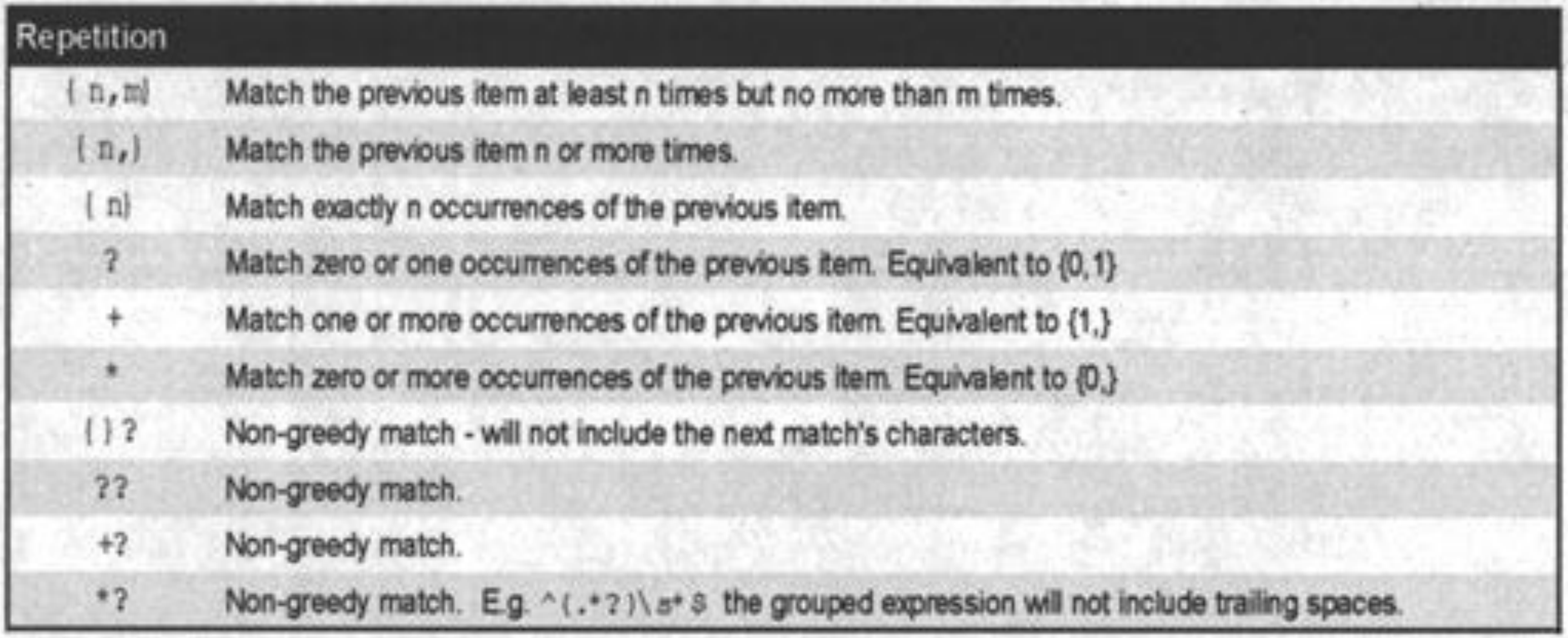
Literal Characters

`\f`	Form feed
`\n`	Newline (Use `\p` in UltraEdit for platform independent line end)
`\r`	Carriage return
`\t`	Tab
`\v`	Vertical tab
`\a`	Alarm (beep)
`\e`	Escape
`\xxx`	The ASCII character specified by the octal number xxx
`\xnn`	The ASCII character specified by the hexadecimal number nn
`\cX`	The control character ^X. For example, \cI is equivalent to \t and \cJ is equivalent to \n

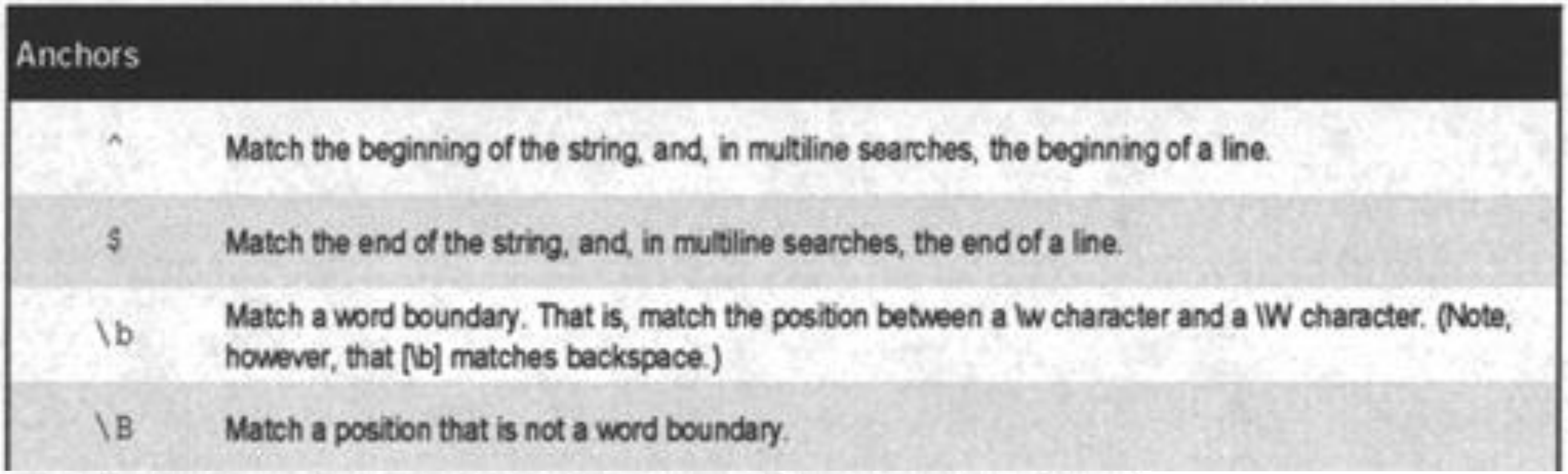
Character Classes

`[...]`	Any one character between the brackets.
`[^...]`	Any one character not between the brackets.
`.`	Any character except newline. Equivalent to [^\n]
`\w`	Any word character. Equivalent to `[a-zA-Z0-9_]` and `[[:alnum:]_]`
`\W`	Any non-word character. Equivalent to `[^a-zA-Z0-9_]` and `[^[:alnum:]_]`
`\s`	Any whitespace character. Equivalent to `[ \t\n\r\f\v]` and `[[:space:]]`
`\S`	Any non-whitespace. Equivalent to `[^ \t\n\r\f\v]` and `[^[:space:]]` Note: \w != \S
`\d`	Any digit. Equivalent to `[0-9]` and `[[:digit:]]`
`\D`	Any character other than a digit. Equivalent to `[^0-9]` and `[^[:digit:]]`
`[\b]`	A literal backspace (special case)
`[[:class:]]`	alnum alpha ascii blank cntrl digit graph lower print punct space upper xdigit

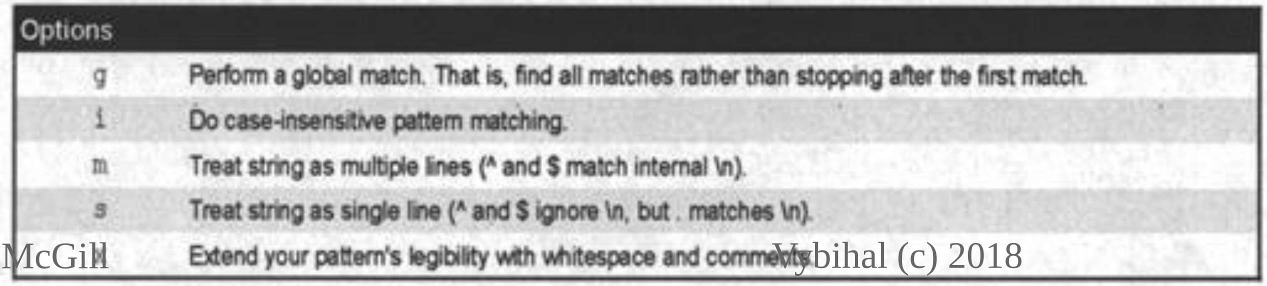
Unix
Bash
C
GNU
Systems



Repetition	
{n,m}	Match the previous item at least n times but no more than m times.
{n,}	Match the previous item n or more times.
{n}	Match exactly n occurrences of the previous item.
?	Match zero or one occurrences of the previous item. Equivalent to {0,1}
+	Match one or more occurrences of the previous item. Equivalent to {1,}
*	Match zero or more occurrences of the previous item. Equivalent to {0,}
{}?	Non-greedy match - will not include the next match's characters.
??	Non-greedy match.
+?	Non-greedy match.
*?	Non-greedy match. E.g. ^(.*?)\s*$ the grouped expression will not include trailing spaces.



Anchors	
^	Match the beginning of the string, and, in multiline searches, the beginning of a line.
$	Match the end of the string, and, in multiline searches, the end of a line.
\b	Match a word boundary. That is, match the position between a \w character and a \W character. (Note, however, that [\b] matches backspace.)
\B	Match a position that is not a word boundary.



Options	
g	Perform a global match. That is, find all matches rather than stopping after the first match.
i	Do case-insensitive pattern matching.
m	Treat string as multiple lines (^ and $ match internal \n).
s	Treat string as single line (^ and $ ignore \n, but . matches \n).
x	Extend your pattern's legibility with whitespace and comments.

When to use grep

- Grep is a useful tool to find specific strings.
  - Outlining all the errors in a log file.
  - Finding a specific string in a collection of source files.
- It becomes an even more powerful tool when combined with other utilities.

```
[jvybihal][~/cs206] who | grep ’mar*’
  mary
  mary ann
  marigold
```

Unix
Bash
C
GNU
Systems

COMP 206 – Joseph Vybihal
Software Systems

Redirection

The ability to send the output from one program into the input of another program

Reading: http://ryanstutorials.net/linuxtutorial/piping.php

Redirection

“send somewhere else”

- Normal output goes to the screen.
  - AKA: STDOUT “standard out”
- Output sent to the screen can be redirected.
  - Symbol: > redirect from screen to a file
  - Ex: ls -la > list.txt
  - Symbol: >> redirect from screen append to existing file
  - Ex: ls -la >> list.txt
  - Symbol: | output from one program sent as input to another program
    Ex: cat test.txt sample.txt | more
- Input from file can be redirected (as is from keyboard)
  - Symbol: < contents of a file sent as input into the program
    Ex: myprogram < input.txt > output.txt

```
$ cat letter.doc > abc.txt
```

```
$ gcc f1.c
Error
Error
Error
```

vs

```
$ gcc f1.c > error.txt
$ more error.txt
```

```
$ cat letter.doc mary.doc /jack/source/backup/stuff.doc > abc.txt
```

vs

```
$ cat /jack/source/letter.txt /mary/source/abc.txt
```

Examples

- The previous commands can easily be combined with the ls command.

- `ls -la | more` will present a paginate list of files.
- `ls -la | head` will present only the first 10 files.
- `ls -la | tail` will present only the last 10 files.

- `` cat `ls *.log | tail -n5` >> text.out ``
concatenate the last 5 log files in the current directory and write them to the text.out file.

Nested execute symbol (backwards quote)

```
$ ls

F1.log
F2.log
:
F20.log

$ cat `ls *.log | tail -n5` >> text.out

$ cat text.out

Contents of: f16.log f17.log... f20.log
```

```
$ ls
F1 f2 f3

$ ls > text

$ ls | more

$ ls *.txt | tail

$ tail *.txt | cat > merged.txt
```

Part of POSIX

File Descriptors

- A file descriptor is created by the OS when a file is opened. The descriptor is the reference to that file.
- Unix has three special file descriptors which are always opened: **STDIN**, **STDOUT** and **STDERR**.

–STDIN 0 (Standard In) : this is the channel were keys typed by the user are gathered.

–STDOUT 1 (Standard Out) : this is the channel were normal application output is sent.

–STDERR 2 (Standard Error) : this is the channel were error output is sent.

- Normal output and error output is separated on two different channels since they are often monitored in different ways.

Part 2

VIM and Developer Techniques

Text File/Source Code Editors

Editors

- Command line text editors allow you to create/edit files at the command line. Several text editors are available.
  - **Vi** or Vim
    - One of the original text editors available on Unix. It's difficult to learn. However, its very powerful and available on every Unix machine.
  - Pico
    - A simple text editor based on the pine mail client. It's very easy to use, and is available on most Unix machines.
  - Emacs
    - Popular and powerful. A heavy weight application.
- You can also use graphical text editors, such as **Vim**, bluefish, gedit or jedit.
- As a long term investment, I highly suggest you learn vi and vim.

Emacs or Vi

- Both command-line editors
- Both very common editors in Unix environments
- Vi > Emacs, in number of environments
- Vi is a light-weight program (needs less system resources)
- Emacs is a heavy-weight program (needs more system resources)
- Both have devoted followers
- Vi is supported on more remote connections
- Emacs has more features

The Vi Editor

Vi's Modes

- Since no menu system, it uses modes (keyboard switch)

- Insert Mode: (ESC i)
  - to edit your text
  - can press any keyboard characters
  - most vi's let you use arrow key
- Escape Mode: (ESC)
  - terminates edit
  - can use arrow keys
  - can use special one letter command
- Command Mode: (ESC :)
  - issue commands like Save, Load, and Quite

Important Commands

- Inserting
  - Any of the following: i, a, o, O
- In ESC mode
  - To delete: dd, x, r
  - To search: /
- Command mode
  - w, q, wq, q!, line number, e filename

A sample Vi session…

Vi Quick Reference

Entering and Leaving vi

% vi *name*	edit *name* at top
% vi +*n name*	... at line *n*
% vi + *name*	... at end
% vi –r	list saved files
% vi –r *name*	recover file *name*
% vi *name* ...	edit first; rest via :n
% vi –t *tag*	start at *tag*
% vi +/*pat name*	search for *pat*
% view *name*	read only mode
ZZ	exit from *vi*, saving changes
CTRL-Z	stop *vi* for later resumption

The Display

Last line	Error messages, echoing input to : / ? and !, feedback about i/o and large changes.
@ lines	On screen only, not in file.
~ lines	Lines past end of file.
CTRL-*x*	Control characters, DEL is delete.
tabs	Expand to spaces, cursor at last.

Vi Modes

Command	Normal and initial state. Others return here. ESC (escape) cancels partial command.
Insert	Entered by a i A I o O c C s S R. Arbitrary text then terminates with ESC character, or abnormally with interrupt.
Last line	Reading input for : / ? or !; terminate with ESC or CR to execute, interrupt to cancel.

Counts Before vi Commands

line/column number	z G |
scroll amount	CTRL-D CTRL-U
replicate insert	a i A I
repeat effect	most rest

Simple Commands

dw	delete a word
de	... leaving punctuation
dd	delete a line
3dd	... 3 lines
i*text*ESC	insert text *abc*
cw*new*ESC	change word to *new*
ea*s*ESC	pluralize word
xp	transpose characters

Interrupting, Cancelling

ESC	end insert or incomplete cmd
CTRL-C	interrupt (or DEL)
CTRL-L	refresh screen if scrambled

File Manipulation

:w	write back changes
:wq	write and quit
:q	quit
:q!	quit, discard changes
:e *name*	edit file *name*
:e!	reedit, discard changes
:e + *name*	edit, starting at end
:e +*n*	edit starting at line *n*
:e #	edit alternate file
CTRL-^	synonym for :e #
:w *name*	write file *name*
:w! *name*	overwrite file *name*
:sh	run shell, then return
:!*cmd*	run *cmd*, then return
:n	edit next file in arglist
:n *args*	specify new arglist
:f	show current file and line
CTRL-G	synonym for :f
:ta *tag*	to tag file entry *tag*
CTRL-]	:ta, following word is *tag*

Positioning within File

CTRL-F	forward screenfull
CTRL-B	backward screenfull
CTRL-D	scroll down half screen
CTRL-U	scroll up half screen
G	goto line (end default)
/*pat*	next line matching *pat*
?*pat*	prev line matching *pat*
n	repeat last / or ?
N	reverse last / or ?
/*pat*/+*n*	n'th line after *pat*
?*pat*?–*n*	n'th line before *pat*
]]	next section/function
[[	previous section/function
%	find matching ( ) { or }

Adjusting the Screen

CTRL-L	clear and redraw
CTRL-R	retype, eliminate @ lines
zCR	redraw, current at window top
z–	... at bottom
z.	... at center
/*pat*/z–	*pat* line at bottom
z*n*.	use *n* line window
CTRL-E	scroll window down 1 line
CTRL-Y	scroll window up 1 line

Marking and Returning

``	previous context
''	... at first non-white in line
m*x*	mark position with letter *x*
`*x*	to mark *x*
'*x*	... at first non-white in line

Line Positioning

H	home window line
L	last window line
M	middle window line
+	next line, at first non-white
–	previous line, at first non-white
CR	return, same as +
↓ or j	next line, same column
↑ or k	previous line, same column

Character Positioning

^	first non-blank
0	beginning of line
$	end of line
h or →	forward
l or ←	backwards
CTRL-H	same as ←
space	same as →
f*x*	find *x* forward
F*x*	f backward
t*x*	upto *x* forward
T*x*	back upto *x*
;	repeat last f F t or T
,	inverse of ;
|	to specified column
%	find matching ({) or }

Words, Sentences, Paragraphs

w	word forward
b	back word
e	end of word
)	to next sentence
}	to next paragraph
(	back sentence
{	back paragraph
W	blank delimited word
B	back W
E	to end of W

Commands for LISP

)	Forward s-expression
}	... but don't stop at atoms
(	Back s-expression
{	... but don't stop at atoms

Corrections During Insert

CTRL-H	erase last character
CTRL-W	erases last word
erase	your erase, same as CTRL-H
kill	your kill, erase input this line
\	escapes CTRL-H, your erase and kill
ESC	ends insertion, back to command
CTRL-C	interrupt, terminates insert
CTRL-D	backtab over *autoindent*
CTRL-^D	kill *autoindent*, save for next
0CTRL-D	... but at margin next also
CTRL-V	quote non-printing character

Insert and Replace

a	append after cursor
i	insert before
A	append at end of line
I	insert before first non-blank
o	open line below
O	open above
r*x*	replace single char with *x*
R	replace characters

Operators (double to affect lines)

d	delete
c	change
<	left shift
>	right shift
!	filter through command
=	indent for LISP
y	yank lines to buffer

Miscellaneous Operations

C	change rest of line
D	delete rest of line
s	substitute chars
S	substitute lines
J	join lirles
x	delete characters
X	... before cursor
Y	yank lines

Yank and Put

p	put back lines
P	put before
"*x*p	put from buffer *x*
"*x*y	yank to buffer *x*
"*x*d	delete into buffer *x*

Undo, Redo, Retrieve

u	undo last change
U	restore current line
.	repeat last change
"*d*p	retrieve *d*'th last delete

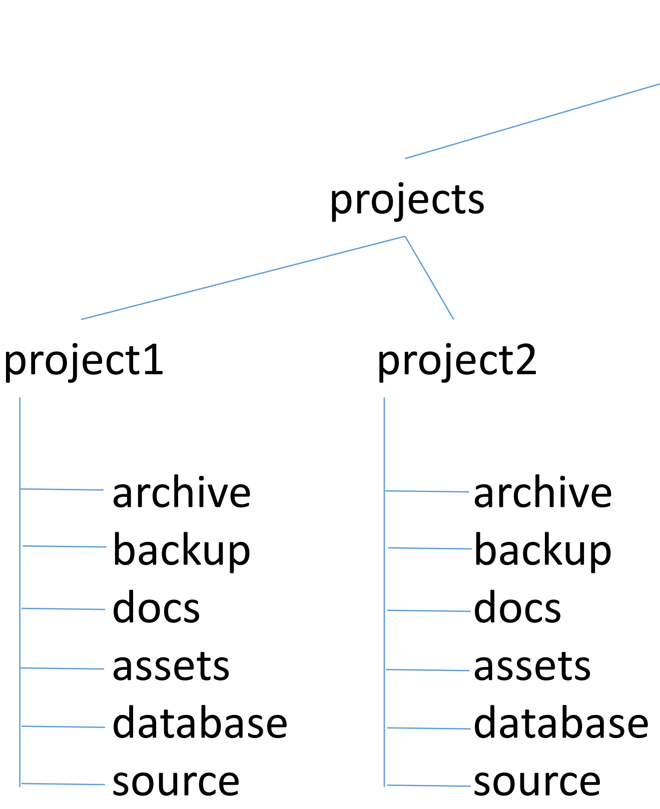
Techniques

- Development techniques
  - Proper filing: common directory structures
  - Common usage procedures
  - File security and sharing
  - Backups and Archiving

Development Techniques

- Important to manage your system resources properly
  - Eg: File management, directories, disk space, nomenclature
  - Learning from others, teach others, evolve
  - Find a good way and stick with it
- Definition of good
  - Low system requirements
    - Your usage of the computer system should practice the Zen technique of limiting system resource impact (memory, CPU, connected devices)
  - Useful qualities in goodness
    - Fast processes
    - Keep things simple

Developer’s Directory Structure



home

projects ….

project1 project2

archive
backup
docs
assets
database
source

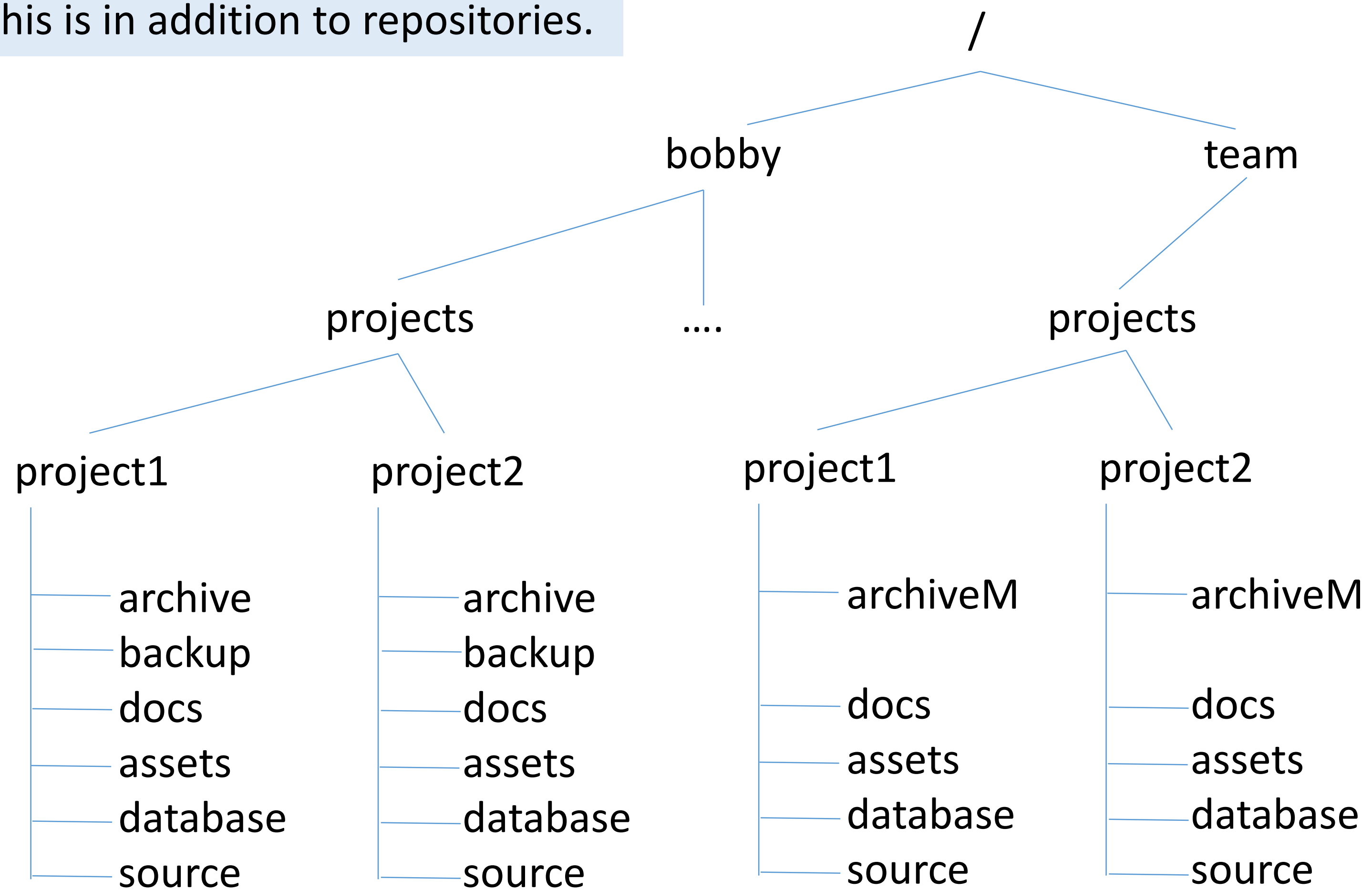
archive
backup
docs
assets
database
source

This is in addition to <u>repositories</u>.

Archive – history backups of all stable versions
Backup – temporary copy of current version
Docs – reading material related to the project
Assets – images, sounds, video
Database – all data saved/read by the program
Source – current source code of the project

Team Directory Structure

This is in addition to repositories.



/

bobby

team

projects

....

projects

project1

project2

project1

project2

archive
backup
docs
assets
database
source

archive
backup
docs
assets
database
source

archiveM
docs
assets
database
source

archiveM
docs
assets
database
source

Computers are not the same…

Notice no Backup directory and the archiveM directory (archive master).
Each member develops on their own with a local copy of the master source & etc.
Once they think they are done they must copy changes to master and test.
Master is the “official” version of the project.

Common Usage Procedures

- At log in
  - Write scripts to help you get to where you want to go
  - Write scripts to customize the environment
- During development
  - Write scripts that help you to
    - Compile quickly and manage errors and executing the program
    - Copying to and from master project
    - Making your own local backups
- At logout
  - Write scripts to do housekeeping
    - Automating backup procedures
    - Automating the logging of events
    - Automating the deletion of files (empty trash)

What commands might we use?

(Not asking about scripts)

(asking about command line)

Common Developer Commands

Unix
Bash
C
GNU
Systems

Archives

- TAR, GZIP, GUNZIP
- An archive is a collection of files combined into one file.
  - Being one file, archives are easier to manipulate (move, store, copy, backup, etc).
  - Archives are often compressed, so they require less space.
- The two most command archive tools used on Unix systems is tar and gzip (gunzip).
  - Tar allows you to combine several files into a single file.
  - Gzip allows you to compress a single file.
  - To compress a collection of files, you need to use both tar and gzip.
- Other archive tools are available.
  - Zip, bzip2, 7z, rar, arj, etc

COMP 206 – Joseph Vybihal
Software Systems

Tar

- Allows the manipulation (creation, extraction) of archive files.
  - A file ending with the .tar extension is a tar archive file.
  - A file ending with the .tgz extension is a compressed (gzipped) tar archive file.
- Switches:
  - -c : create a new tar archive
  - -r : update the tar archive
  - -x : extract from the tar archive
  - -f: specifies the archive file name.
  - -v: activates verbose mode, which means the tar command will output lots of information.
  - -z: allows you to compress the archive (the archive is compress/decompressed using gzip).

Tar (cont.)

- Here are a few example of the tar command:

  - tar -cvf log.tar *.log
  - tar -zcvf log.tgz *.log
  - tar -xvf log.tar /tmp/log
  - tar -zxvf log.tgz /tmp/log

- The first two commands create an archive with log files. (one normal and one compress)
- The two following commands show how to extract those two archive.

DIFF

- The comparison of two files
  - Developers use this command to help them find out if two source files are the same or what was changed in a source file.
    - When working in a team it is common that one developer changes a file someone else did not want changed.
    - Or, the team leader would like to know how much work was done on a file.
- diff [options] file1 file2

ln : Hard and Symbolic Links

- LN and LN -S
- The ln command can be used to create links to files and folders.
  - Hard link: ln link_name /path/file
  - Soft link: ln –s link_name /path/file

- When creating a hard link, you are simply giving another name to a file (it shows up separately on an ls – a direct pointer in directory)
  - The link will point to the same physical space on the disk.
  - A file can only be deleted once all its hard link are deleted.

- When creating a symbolic link (using ln -s), a new file is created (an indirect pointer in directory)
  - The new file automatically redirects to the target file.
  - Symbolic links can be created across volumes (or disks).
  - Deleting a symbolic link does not affect the target file.

More Commands

- sort [options] file
  - sort the lines of the file

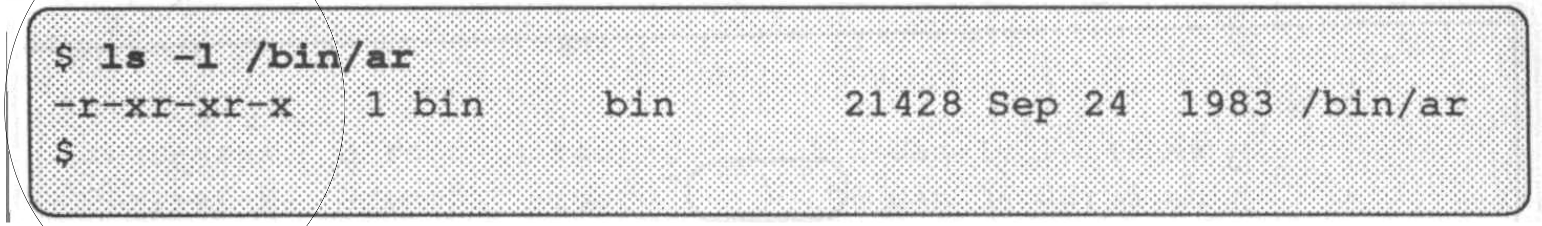
- touch [options] [date] file
  - create an empty file, or update the access time

- wc [options] [file(s)]
  - display a count of words (or character or line)

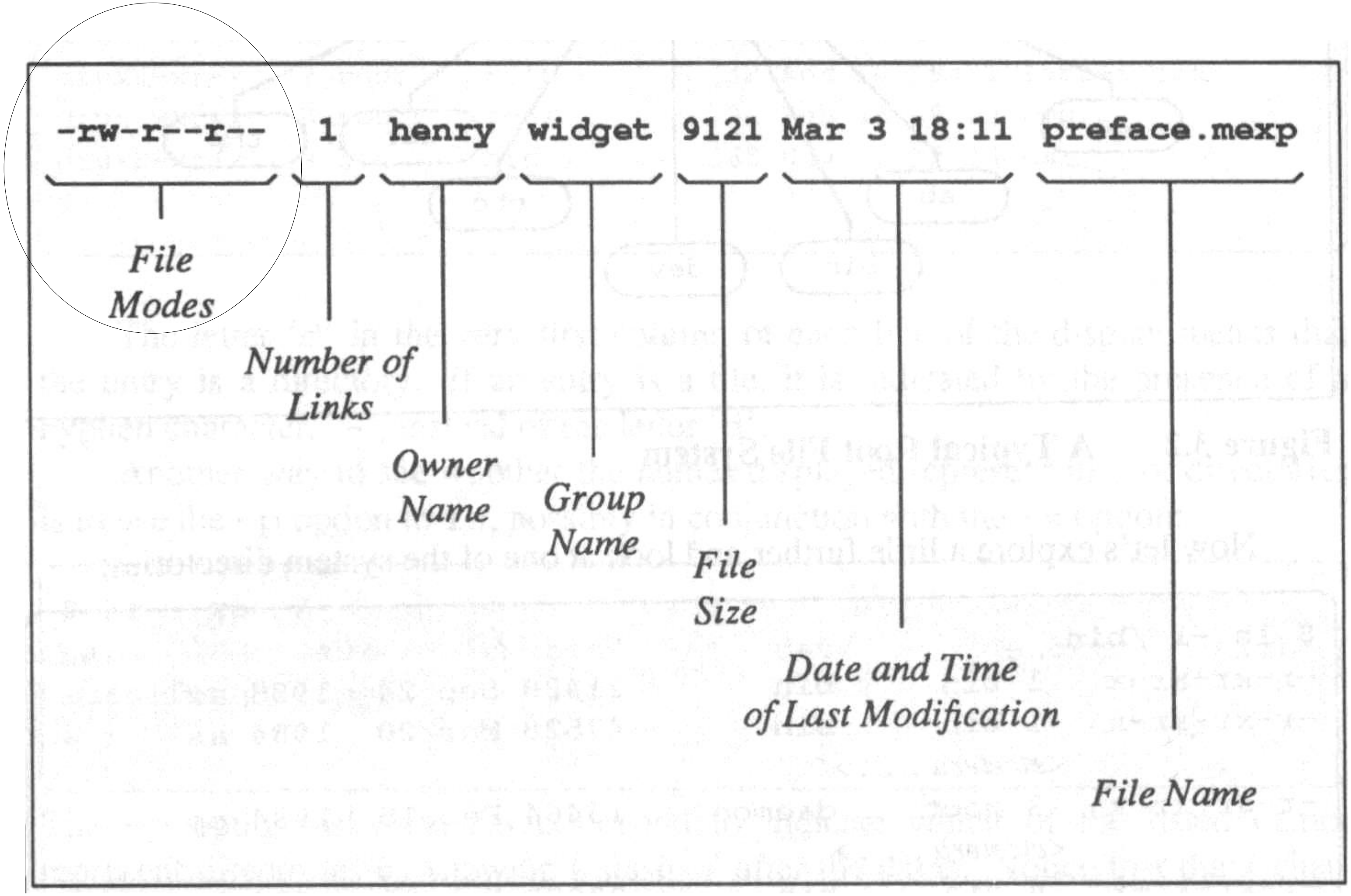
Security

Permissions on the File Systems

- All files are **owned** by a user and a group.
  - Usually, this owner is the user that created the file.
- Permissions on files exists at three level: **user**, **group** and **all**.
- Three types of rights can be given: **read**, **write** and **execute**.
- Any combination of these rights must be given to these three levels.



```
$ ls -l /bin/ar
-r-xr-xr-x   1 bin     bin         21428 Sep 24  1983 /bin/ar
$
```



```
-rw-r--r--  1  henry  widget  9121 Mar 3 18:11  preface.mexp
```

File Modes

Number of Links

Owner Name

Group Name

File Size

Date and Time of Last Modification

File Name

Permissions (cont.)

d,rwx,rwx,rwx

- Permissions are displayed as a string of 10 characters
  - 1st : indicates if the file is a directory.
  - 2nd : indicates if the owner has read access to the file.
  - 3rd : indicates if the owner has write access to the file.
  - 4th : indicates if the owner has execute access to the file.
  - 5th , 6th , 7th : indicates if the group owner has read, write or execute.
  - 8th , 9th, 10th : indicates if all other users have read, write or execute.

Do permissions overlap?

- Given the permission “------rwx“ of a file I own, can I read the file?
  - You will not be able to read the file.
  - People in the group will not be able to read the file.
  - Other people will be able to read the file.

Note: some Unix systems interpret Other as All... this changes things.

Quiz: Can I read, write, execute?

Can user “Bob” of group “Student” read, write or execute the following files?

rwxr--r--	Cathy	Frosh	file1.sh
r-x------	John	Student	file2.txt
rwxrwxr--	Bell	Student	file3.txt
rwxrwxrwx	George	Teacher	file4.c
rwx------	Bob	Student	file5.s
rw-rw-r-x	Norm	Admin	file6.doc
rwxrwx---	all	all	file7
------rwx	Bob	Student	file8.doc
---rwx---	Bob	Student	file9.txt

Bonus: Which write does root have on these files?

CHMOD – change mode

- The chmod command is used to change permissions:
  - Who:
    - u : The user who owns the file (this means “you.”)
    - g : The group the file belongs to.
    - o : The other users
    - a : all of the above (an abbreviation for ugo)
  - Permission
    - r : Permission to read the file.
    - w : Permission to write (or delete) the file.
    - x : Permission to execute the file, or, in the case of a directory, search it.
  - Changes to
    - = : become
    - + : add
    - - : remove

Examples

- The syntax of the command is as follows:
  - `chmod who=permission files`
- Here are a few examples of the chmod command:
  - Give read permission to group
  - `chmod g+r file.txt`
- Give read/write/execute permission to you (user)
  - `chmod u+wx file2.txt     - rwx --- ---`
- Remove all permissions from others
  - `chmod o= file3.txt`
- Give read/write permission to user and group
  - `chmod ug=rw file4.* file2.txt`

Binary Settings

000	0
001	1
010	2

- Bit Setting:
- rwx rwx rwx   in symbolic form
- 111 000 111   in bit form (1=on, 0=off)
- 707   in base 10 version of bits

- chmod 707 *.doc
- rwx for owner and other, but not for group

Unix
Bash
C
GNU
Systems

COMP 206 – Joseph Vybihal
Software Systems

Binary

000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7
rwx	

rw-	--x	---
110	001	000
6	1	0

chmod 610 filename

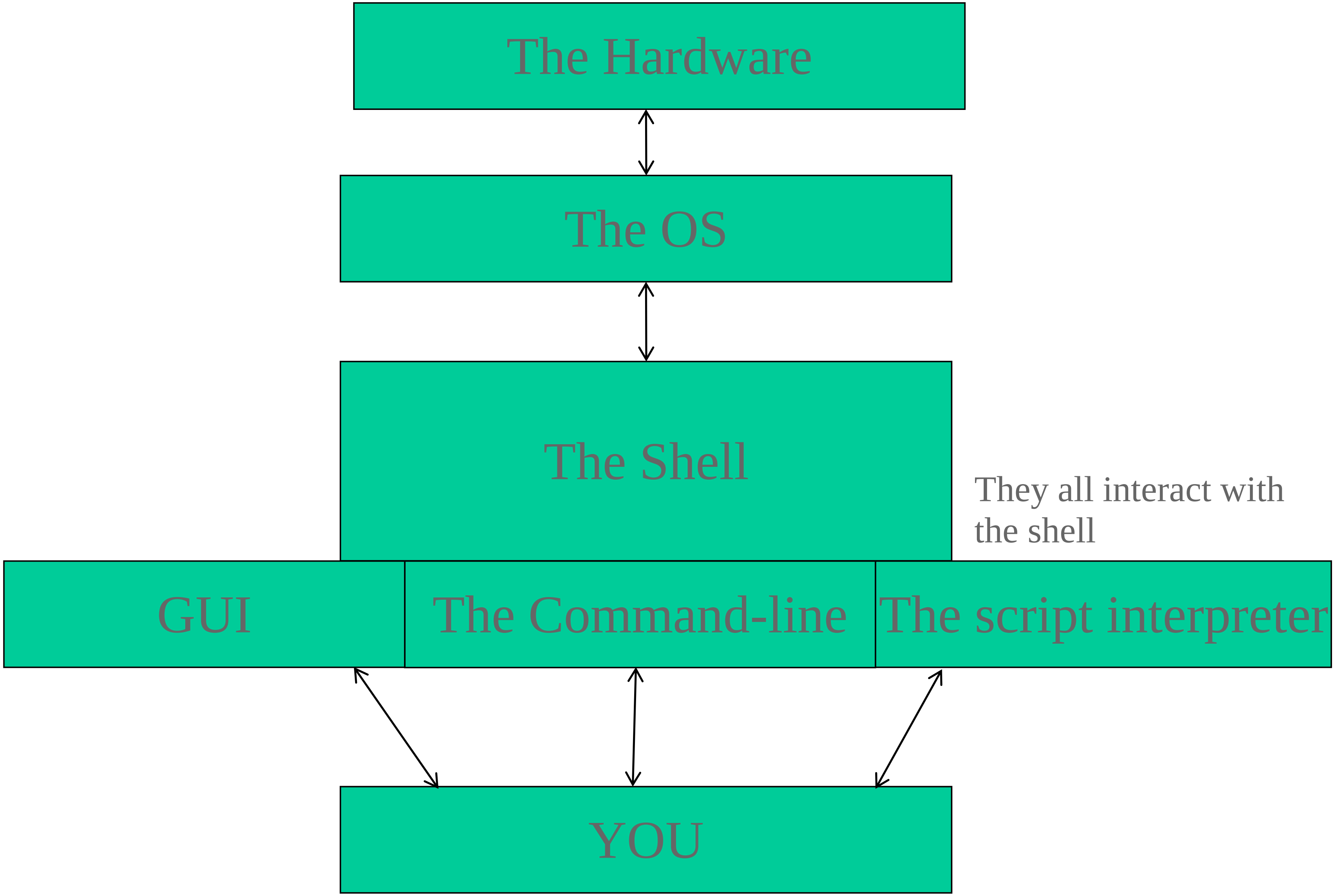
Part 3

Intro to Bash Scripting

Readings

- Chapter 2 from textbook
- Bash and command-line help:
  - http://tldp.org/HOWTO/Bash-Prog-Intro-HOWTO.html
  - http://ss64.com/bash/

The Architecture



They all interact with the shell

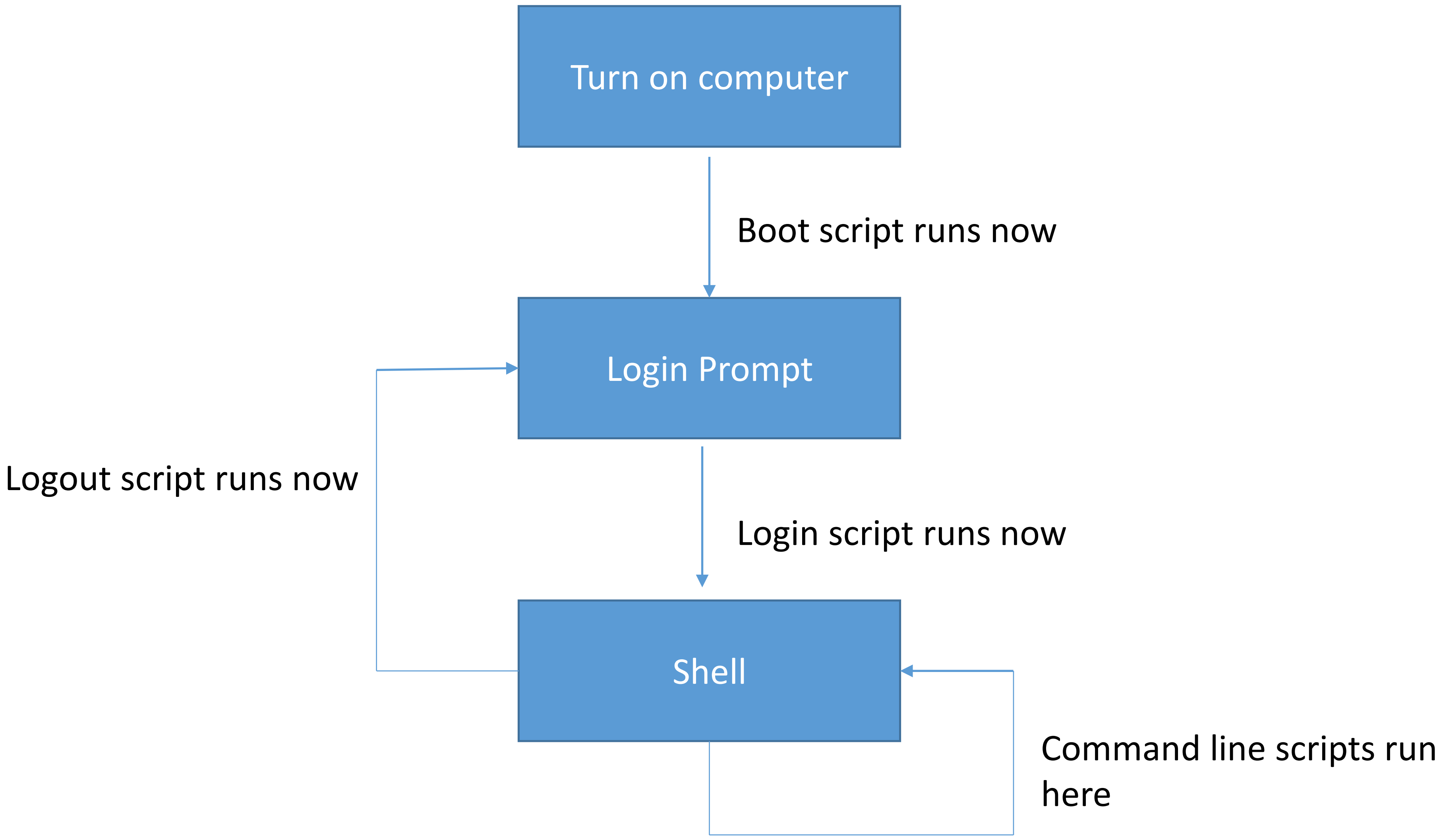
Scripts are used here

- At log in
  - Write scripts to help you get to where you want to go
  - Write scripts to customize the environment
- During development
  - Write scripts that help you to
    - Compile quickly and manage errors and executing the program
    - Copying to and from master
    - Making your own local backups
- At logout
  - Write scripts to do housekeeping
    - Automating backup procedures
    - Automating the logging of events
    - Automating the deletion of files (empty trash)

Two kinds of scripts

- Boot / Login scripts
  - Used to modify the OS environment
  - Boot scripts
    - created by super user for all users
  - Login scripts
    - created by account owner for account
- Command-line scripts
  - Created by users to automate command-line activities

Two Kinds of Scripts



Scripts

- Scripts are collections of commands, grouped in a file and sequentially execute.
- Scripts are not compiled programs, they are interpreted.
- Scripts run from the top to bottom of the file
- It must be CHMOD'd to execute

Bash

- BASH is a Unix scripting language that implements some programming language control flows, like:
  - functions
  - if
  - for
  - while
- It is interpreted by the OS, not compiled for the CPU.

Example script with demo

```
clear
who
finger bob
ie
outlook
quoteoftheday
```

A script

```
chmod +x file
```

Make text file executable

```
./file
```

Running the script

Remote Issues

clear

who

finger bob

ie

outlook

Cannot do through ssh

quoteoftheday

Only if app was installed

```
$ vi bashfile
$ chmod +x bashfile
$ ./bashfile
```

The sha-bang

- The sha-bang #!
  - The first line of the script should start with **#! PROGRAM**
  - Indicates to the OS that the script is to be executed by PROGRAM
    - Different languages can be used to script (sh, bash, perl, python, ruby, etc).
- To set up a Bourne shell script the first line must be:
  - #!/bin/sh
- To set up a Bash shell script the first line must be:
  - #!/bin/bash

Example

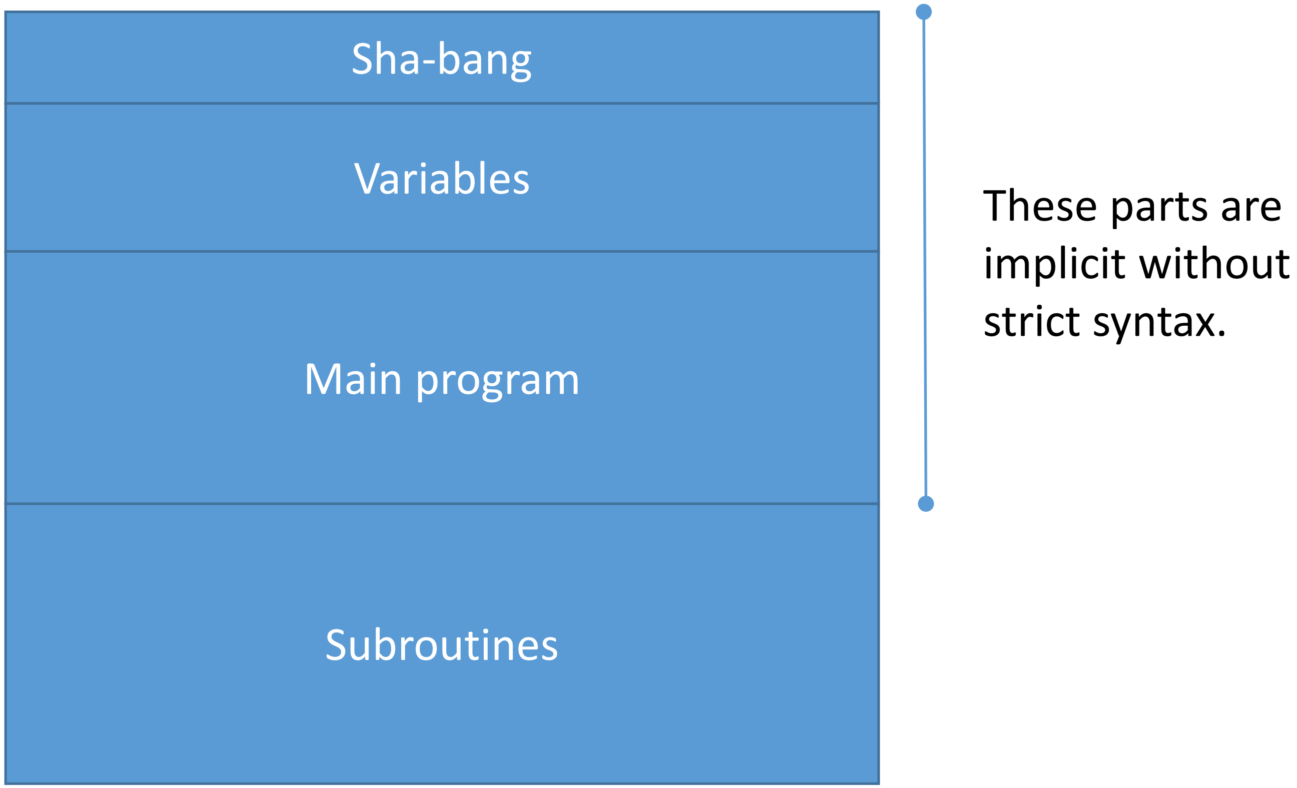
```
#!/bin/bash
clear
who
finger bob
quoteoftheday
```

Declaring that the script interpreter must be Bash

If you do not provide a sha-bang then it will default to the sh shell.

```
$ vi bashfile
$ chmod +x bashfile
$ ./bashfile
```

Bash File Structure



These parts are implicit without strict syntax.

The above organization is customary, however it is not required to be written in this manner. Variables can be declared anywhere and subroutines do not need to be defined at the end of the file.

Bash File Structure

```
#!/bin/bash
clear
who
finger bob
quoteoftheday
```

The sha-bang

The main program

Notice how the script does not have strong syntax requirements. For example the main program does not have a begin-end syntax.

Example

```
$ vi backup.sh
```

```
#! /bin/bash

# This is a comment
# Backup files, remove and verify

cp *.txt /home/jack/backup
rm *.txt
ls *.txt
```

```
$ chmod +x backup.sh
$ ./backup.sh
```

Example

```
$ vi morningRoutine.sh
```

```
#! /bin/bash

# What I like to do each morning

who
chrome http://mail.cs.mcgill.ca
date > today
time >> today
weather >> today
cat today
```

```
$ chmod +x morningRoutine.sh
$ ./morningRoutine.sh
```

Example

```
$ vi search
```

```
#! /bin/bash

# Find a file

grep $1 `ls`          # searches within the files for $1
ls | grep $1          # compares the file name for $1
```

```
$ chmod +x search
$ ./search dog
```

This becomes $1