Software skills for librarians: Library carpentry

Module 1: The Unix shell and regular expressions

Introduction to computers

- What the computer is:

  A useful tool

  Obedient

  Accurate & Fast

- And what it is not:

  An electronic brain

  Intelligent

  Magic

What computers do well

- Simple, repetitive tasks
- Follow instructions
- Number crunching
- Look for patterns in regular data
- Remember and process large data sets

Why program

- Speed up repetitive tasks
- Professional development for yourself and others
- Help understand other automation projects
- Curiosity

Main lessons

- Borrow and reuse:

  Look at other's code

  Use libraries

  Re-use sections of your own code

- There is no best language:

  They are designed for different tasks

  Each has strengths and weaknesses

  Same fundamental principles

Computational thinking

- Abstraction:

  Transistors, Boolean logic, Machine code, Programs

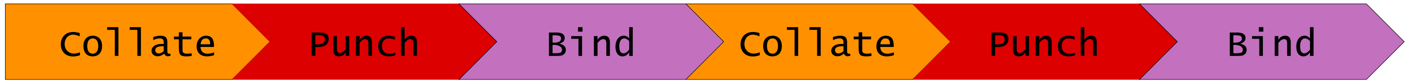
- Helps to handle complexity
- Black boxes:

  Inputs transformed into outputs

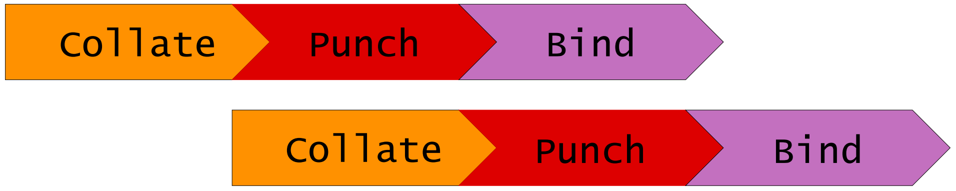
- Think about the process

Workflows

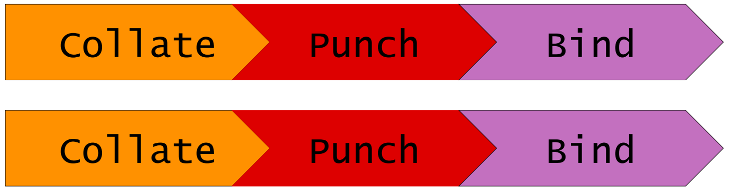
- Sequential



- Pipelined



- Parallel



Introduction to the unix shell

- Interact with users
- Command line interface
- Read Evaluate Print loop
- Disadvantages:

  Terse, cryptic commands, text only
- Advantages:

  Faster, easier to automate, easier to program

Example

- Report on MARC field usage
- Single command
- Repeatable:

  Shell history, up arrow key
- Loop over all fields
- Compare with imaginary GUI

Simple commands

- `pwd` – print working directory
- `ls` – list files
- Commands can take parameters

  `ls textfiles` – list contents of directory textfiles
- Commands can take options

  `ls -l` – list files in the ‘long’ format

Directories

- Single directory tree
- Slash at the beginning indicates the root
- In the middle it separates names
- Absolute path from root, relative from current directory
- Filenames can include any character

  Enclose parameters with a space in quotes, eg `"file name"`
- Users files in `/users`

Navigating the filing system

- `cd` – change directory
- `cd ~` – go back to home directory
- `cd ..` – go up to parent directory
- Filename extensions
- Tab completion
- Wildcard expansion:

  *.txt – all text files

  img?.jpg – JPEG images img1, img2 ...

  file[1-9].txt – file1.txt, file2.txt, etc.

Getting help

- `--help` after command name
- no parameters or incorrect syntax
- `man` <command name>
- Google
- Unix in a nutshell

Working with files

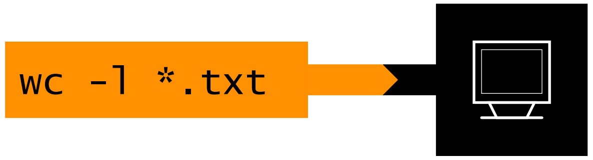
- `mkdir` – create directory
- `nano` – text editor
- `rm` – delete a file
- `cp` – copy a file
- `mv` – move, or rename, a file

Examining files

- `wc` – word count
- `cat` – print a file
- `sort` – sort the lines of a file into order
- `head` – print first n lines
- `tail` – print last n lines
- `diff` – difference between two files

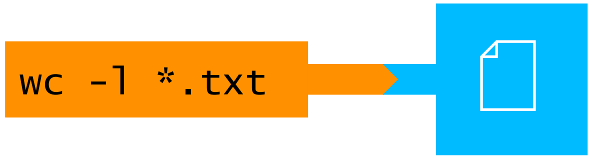
Redirection

- Standard input and output



- Redirect output to a file: `> file`

```
wc -l *.txt > out.txt
```

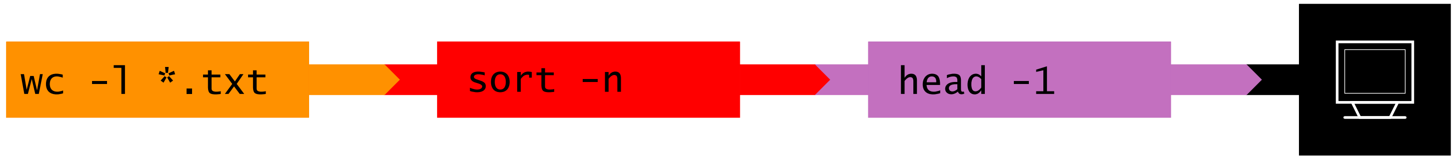


Redirection continued

- Pipe output of one command to input of second

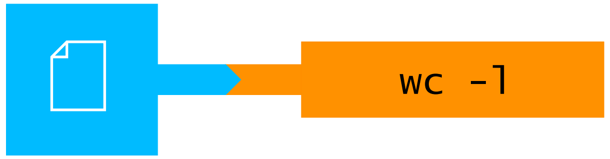
```
first | second
```

```
wc -l *.txt | sort -n | head -1
```



- Read input from a file: `< file`

```
wc -l < in.txt
```



Shell scripts

- Simple programming, save and recall the steps for common tasks
- Variables, labels for pieces of data
- Loops, repeat the same command or operations several times

```
for filename in record.txt record.marc;
do cat $filename;
done
```

- Can use wildcards

Finding text

- grep – search for patterns in text files

```
grep Linux record.txt
grep -w Linux record.txt
grep -n "Addison Wesley" record.txt
grep -E '^650' record.txt
```

Transliterating characters

- Like global search and replace on single characters

  Change every occurance of one character into corresponding one

  Or delete every single character of one type

- Examples of `tr` command:

```
tr 'a-f' 'A-F'
tr '[:upper:]' '[:lower:]'
tr '\012' ' '
tr -d '[:punct:]'
```

Automated editing

- sed -e command filename – perform editing command on each line

```
sed -e 's/^650/655/' record.txt
sed -e '/^650/p' record.txt
sed -e '/^035/d' record.txt
sed -e '1,3d' record.txt
sed -e '1,3p' record.txt
sed -e '1i(UkCU-COM)' record.txt
sed -e '/^020/a$q pbk.' record.txt
```

Regular expressions

- Used to match patterns in text
- Useful for understanding your data
- Or specifying its format
- Similar to search and replace
- Supported by many tools, like `grep` and `sed`

Example

- Organise
- Organize|Organise
- \b(Organize|Organise)\b
- \b[Oo]rgani[sz]e\b

Regular expression syntax

- Most characters match themselves
- Vertical bar for alternatives
- Square brackets for character class
- Round brackets for grouping a subexpression
- \b for word boundaries

Character classes

- Matches any one of the characters in brackets
- [abc] any one of a, b or c
- Could be written as [a-c]
- [A-Za-z] any upper or lower case letter
- [^A-Za-z] anything except a letter
- \w \d \s shortcuts
- Fullstop matches any character

Anchors and back references

- $ Matches only at end of string
- ^ Matches only at beginning
- Adding a slash in front of a special character matches that single character. Eg \$[a-z]
- Brackets have two meanings, grouping and capturing
- \1 refers back to first set of brackets

Repetition

- ? 0 or 1

  Organised?

- * 0 or more

  \$[a-z]([^$.]*)

- + 1 or more

  0x[0-9A-F]+

- {n,m} Between n and m

  \d{10,13}

Further reading

- Unix in a nutshell / Arnold Robbins — 4th ed.

  O'Reilly, 2005 — ISBN 0596100299

- Classic shell scripting / Arnold Robbins and Nelson Beebe.

  O'Reilly, 2005 — ISBN 9780596005955

- Mastering regular expressions / Jeffrey Friedl — 3rd ed.

  O'Reilly, 2006 — ISBN 9780596528126