COMP1005 Week 10 Cheat Sheet

Lisa Luff

10/20/2020

Contents

Workflow

- ASAP
    - Automation
        - Automate computation aspects
        - Repetitive pipelines, sweep campaigns
    - Scaling
        - Compute cycles
        - Makes use of computational infrastructure
        - Handle large data
    - Abstraction
        - People cycles
        - Shield complexity and incompatibilities
        - Report, re-use, evole, share, compare
        - Repeat, tweak, repeat
        - First class commodities
        - FAIR(R): Findable, Accessible, Interopable, Reuseable, (Reproduceable)
    - Provenance
        - Reporting
        - Capture, report and utilise log and data lineage auto-documentation
        - Traceable evolution, audit, transparency
        - Compare
- Workflow categories
    - Instrument pipeline +
    - Data wrangling and analytics +
    - Simulations
- If workflow goes in a straight line, it’s also a pipeline

Unix Power Tools

- Piping
  - Allows you to connect several commands together
  - Output of one, becomes input to the next
  - Most Unix commands get input from stdin, and pass output to stdout
  - “|” The pipe symbol directs Unix to connect stdout from the first command to stdin of the second command
  - > Will redirect the output to a file
  - » will append to an existing file
  - < will redirect input from a file
- Workflow
  - wget - get files from web
  - wc - word count, three numbers are lines, words, characters
  - grep
    - See next page for more
  - awk - F“*field separator*”‘{*do thing a, b, c*}’*filename.filetype* filters a file by fields
    - Reads through file
    - Separates into fields using “,” to separate
    - Counts fields from **1**
    - Prints fields *a, b, c*
    - Could also use cut
  - gnuplot - allows command line plotting
  - head/tail -n - first/last 10 or n lines
  - more/less - scroll through file one page at a time, space on ‘n’ to continue, ‘b’ for backwards, or ‘q’ to exit
- Plotting
  - gnuplot *filename.filetype* - Plots based on plotting commands in the file
  - In file, example: plot for [col=1:4]‘./*file.csv*’using 0:col with lines
    - Plots 4 lines from columns 1 to 4
    - x values use defaults
    - can add labels, titles, etc

Grep

- Allows sophisticated searches using regular expressions with commands

Command	Purpose
grep *option pattern(string) path*	Print only the lines in *path* that match *pattern*
path	starts with / and can use ../ to look in parent directory
$(a \vert b)$	For multiple arguments
command path | grep *pattern*	To use commands with grep
-r or -R will look in directory and all subdirectories	
-i	Not case sensitive
-c	Counts occurances
-w	Will find only exact
-W	Same but for words
-n	Line number found on
-B	Line before found
-A	Line after found
-h	Suppress file names
-*x**y*	How to combine options
–color *option*	Can change colours of output
- P	An option that allows the use of regular expressions
:———	:———
- P command	Use
:———	:———
-v	As option for inverse
.thing	Can be 0 or 1 *things*
.*	Wild card
.{n, m}	Can be between n and m random assortment of things
ˆ	Start of line
$	End of line

Bash Scripts

- Bundle repeated commands into scripts
- Do the things, then save x lines to a file
  - history -*x* > *filename.filetype*
- Or write with vim *filename*.sh
- Customise scripts with command line arguments
  - Can set it up so when you run the program, you also can set arguments as you call it
  - Use $*n* for setting command arguments, separated by a space
  - bash *filename.sh thing other*
    - $1 refers to the file name (first thing after bash)
    - $2 refers to *thing*
    - $3 refers to *other*
    - $@ is all command line arguments
    - In script need “$1”, but in command line, no "" needed
  - Eg, to make a script that collects the middle lines in data
    - head - n “$2” “$1” (collect section) | tail -n “$3” (collect the tail of the section collected by head)
- For loop
  - for *item* in “$@” (example)
    > do
    > > *thing*
    > done

Python Modules and Scripts

- Packages
  - Group modules together
  - ___init___.py indicates a directory is a package
    - Modules are then held within that directory
  - To call on them
    - import *packagename.modulename*

System Calls

- You can work with directories within a program
- OS module (import os) -
  - mkdir(*string*)
  - listdir()
  - chdir(*string*)
  - getcwd()
  - rename(*source*, *destination*)

Command Line Arguments Python

- import sys
  - sys.argv is a list of all command line arguments
  - Use sys.argv[*n*] to use the nth argument entered

Parameter Sweeps

- Used for:
  - Finding optimum value of a parameter
  - For studying the sensitivity of the design performance to certain parameters
  - Running a series of simulations with a set of varying parameters
- Loops through all permutation of the values
- Analyse the results after loops are complete
- How to use:
  - Can be linear or logspace values
  - May be string values in a list
  - Good to have this part controlled through input files or command line arguments
  - Can call a python script from a driver bash script to give the parameter sweep
- For data management:
  - Scripts to automate experiments
  - Use additional scripts to do multiple runs
  - Create directory structure for each experiment and copy supporting files
  - Use date and other meaningful information in directory names
  - Bundle results for each stage of work matching "bundle" for code

Regular Expressions

- regex101.com - good for these
- Regular expressions are for when you're reading files where the formatting isn't consistent
  - Cleans it up without having to do it manually
- You might want to do matching that is more flexible
- Metacharacters
  - Most letters and characters match to themselves
  - Metacharacters have special meaning
    - .^$*+?{}[]\|()
  - [] is a set of characters to match - [cbm]at - will match to cat, bat and mat
  - ^ gives the opposite (complement) - [^5] - will give everything except 5
  - \ gives special sequences, or overrides a metacharacter
  - \d - matches to any decimal digit == [0-9]
  - \D - matches to any non-decimal digit [0^9]
  - \s - matches any whitespace character [\t\n\r\f\v]
  - \S - matches any non-whitespace character
  - \w - matches any alphanumeric character [a-z, A-Z, 0-9]
  - \W - matches any non-alphanumerica character
  - . - matches anything other than newline
  - * - matches to zero or more repeats of the previous colour or class - ca*t matches to ct, cat, caat, caaat, etc.
  - + - matches to one or more repeats - ca+t matches to cat, caat, caaat, etc. but not ct
  - {*m*, *n*} - matches at least *m* repeats and at most *n* repeats - a/{1, 3}b matches to a/b, a//b, a///b, but not ab or a////b, missing m or n defaults to 0 or infinity respectively
  - Can combine special character arguments, eg.
    - .at - matches anything with a character followed by "at"
    - [0-9][a-z] - matches any digit followed by a lowercase character
    - [0-9]\s[a-z] - matches any digit followed by a whitespace character, followed by a lower case character

- Using regular expressions
  - import re
  - Use methods, with r’*thingtosearch*’ as the argument
- Methods
  - *variable* = re.compile() - the expression to use in following functions (if being used multiple times)
    - Then use the pattern in other methods with *variable.function*()
    - Can use r’ ”‘(*expression*) (*other*)” ’, re.VERBOSE) to improve readability of long pattern expressions
  - match() - match to beginning of string
  - search() - match to anywhere in the string
  - findall() - returns a list of matches
  - finditer() - returns an iterator of matches
  - These all output objects, and then we use functions on the objects
    - group() - the string that was matched
    - start() - starting position of the match
    - end() - ending position of the match
    - span() - a tuple containing start and end