Scripting
Overview

• Combining commands
• redirection
• String processing
  – Sed
  – Awk
  – grep
• Cut and paste
Scripting languages

- Scripting language are not compiled like java or c.
- They are interpreted line by line.
- So only have runtime errors, not compilation errors.
- Have been adopted by the big data community for gluing purposed.
- Generally easy to learn, and used for short programs.
- Described as “dynamic high-level general purpose languages”
- E.g. perl, python, bash
Connecting commands

• && (logical and, like java – short circuit)
• ; (serial)
• | (pipe – creates a pipeline of commands)
• Think about the difference between these.
&&

- && is the Boolean operator
- `javac prog.java && java prog`
- This will only execute the second command if the first command is successful.
;  

- Command1; command2
- This is useful for connecting two commands
• This is a very useful way of connecting commands.
• The output of command1 is the input to command2
• command1 | command2
example 1

- `jrw@Tambala ~/zzTrash`
- `$ ls -l | grep ^d`
- `drwxr-xr-x+ 1 jrw Domain Users 0 Apr 29  2014 colin`
- `drwxr-xr-x+ 1 jrw Domain Users 0 Mar  9 16:47 dir1`
- This command only lists directories.
example 2

- `jrw@Tambala ~/zzTrash`
- `$ ls -l | grep ^\-$
- `-rw-r--r--  1 jrw Domain Users 0 Mar  9 16:47 hi.txt`
- `-rw-r--r--  1 jrw Domain Users 6 Mar  9 16:43 out.txt`
- This command will only list files.
Cut and Paste
Cut c option

1. cut -c5 test.data
2. prints out the 5th column of the file.
3. it does not cut it out.
4. cut -c5-10 test.data
5. prints cols 5 to 10
6. cut -c5,7,10 test.data
7. prints cols 5 7 and 10.
8. if no input file - reads from input.
1. $who | cut -c1-8
2. will tell me who is logged on (usernames).
3. $who | cut -c1-8 | sort
4. will sort the list
5. who | cut -c10-16
6. will tell me what terminals are being used.
7. $who | cut -c1-8,18-
8. will display user name and login time.
9. (these fields are usually this wide)
cut -d –f options

• cut -ddchar -ffield file
• dchar is the delimiting character.
• field is the field
• cut -d: -f1 /etc/passwd
• will set delimiter to : and get the first field
output

- cut -d: -f1 /etc/passwd
- root
- rootnir
- bin
- daemon
- (f1,6 is field 1 and 6)
- the default delimiter is tab
- how would you set the delimiter to space?
Paste

1. Paste is almost the inverse of cut
2. two files names.txt nums.txt
3. paste names.txt nums.txt > namesAndNumbers.txt
4. will paste them side by side!
5. this like not
6. cat names.txt nums.txt > namesAndNumbers.txt
7. which will put one file after another.
Paste -d option

- paste -d, file1 file2
- will use “,” as a separator (delimiter)
- It is best to place in single quotes
- paste -d',' file1 file2
Paste -s option (one file)

- cat names.txt
- john
- dave
- steve
- $paste -s names.txt
- john    dave    steve
Paste -s option (two files)

• $paste -s names.txt nums.txt
• john  dave  steve
• 1    2    3
translate characters - standard input.

- `tr x y < namesAndNumbers.txt`
- translated from x to y in file `namesAndNumbers.txt`
- `tr` can be used to produce more readable output.
- `cut -d: -f1,6 /etc/passwd | tr : \t`
- this replaces one delimiter with another
- making it more readable.
Upper to Lower case

• Upper to Lower case
• tr 'A-Z' 'a-z' < names.txt
• will convert upper case to lower case.
tr -s option (squash)

• tr -s ':' '
11'
• this will replace multiple occurrences of ::::
• with a single tab.
• tr -l ' ' ' ' < poem.txt
• will remove multiple spaces
• and replace with single spaces.
tr -d option (delete)

• tr can delete single characters.
• tr -d ' ' < names.txt
• will remove space from names.txt
• can do same with sed
• sed 's/ //g' names.txt
• (s is substitute, g is global)
String Processing Tools

• You can do some very nice string processing with scripting languages.
  • grep
  • Sed
  • Awk
grep
Regular Expressions

- `grep "That" poem.txt` will only find the string "That" in `poem.txt` if it has an upper case 'T' followed by lower case 'hat'

- Regular expressions are much more powerful notation for matching many different text fragments with a single expression – i.e. could wish to find "That", "that", "tHaT", etc.
Regular Expressions (2)

• Search expressions can be very complex and several characters have special meanings
  – to insist that That matches only at the start of the line use `grep "^That" poem.txt`
  – to insist that it matches only at the end use `grep "That$" poem.txt`
  – a dot matches any single character so that `grep "c.t" poem.txt` matches cat, cbt, cct, etc.
Regular Expressions (3)

• Square brackets allow alternatives:
  – `grep "[Tt]hat$" poem.txt`

• An asterisk allows zero or more repetitions of the preceding match
  – `grep "^-*\" poem.txt` for lines with only -'s or empty
  – `grep "^--*\" poem.txt` for lines with only -'s and at least one -
  – `grep "Bengal.*Sumatra" poem.txt` for lines with Bengal followed sometime later by Sumatra

• Many flags to:
  – display only number of matching lines, ignore case, precede each line by its number on the file and so forth
SEARCH PATTERNS (1) 😊

• /The/
• /^The/
• /The$/
• /\$/
• /\$/
• /[Tt]he/
• /[a-z]/
• /[a-zA-Z0-9]/
Patten for a (possibly signed) integer number.

- `/^\+[\-]?[0-9]+$/` -- matches any line that consists only of a (possibly signed) integer number.
- `/^ Find string at beginning of line. `
- `/^[-+]? Specify possible "-" or "+" sign for number. `
- `/^[-+]?[0-9]+ Specify one or more digits "0" through "9". `
- `/^[-+]?[0-9]+$/ Specify that the line ends with the number.`
SED

*Sed* is the ultimate stream **editor**

s for substitution

• sed s/day/night/ oldfile.txt
• This will print to screen
• sed s/day/night/ oldfile.txt >newfile.txt
• This redirects to new file.
• echo day | sed s/day/night/
• This goes to the screen:
substitute command

- There are four parts to this substitute command:
  1. s Substitute command
  2. ../../ Delimiter
  3. day Regular Expression Search Pattern
  4. night Replacement string
Global replacement

• Most Unix utilities work on files,
• reading a line at a time.
• Sed, by default, is the same way.
• sed 's/cat/dog/' data.txt
• This only replaces single occurrence
• sed 's/cat/dog/g' data.txt
• This replaces all (global) occurrences.
Awk – an introduction
An example text file - Coins.txt

• gold 1 1986 USA American Eagle
• gold 1 1908 Austria-Hungary Franz Josef 100 Korona
• silver 10 1981 USA ingot
• gold 1 1984 Switzerland ingot gold 1 1979 RSA Krugerrand
Using awk

- I could then invoke Awk to list all the gold pieces as follows:

- `awk '/gold/' coins.txt`
Selecting Fields

- awk '/gold/ {print $5,$6,$7,$8}' coins.txt
- This yields:
- (ie. Prints out certain columns)
- American Eagle
- Franz Josef 100 Korona
- ingot Krugerrand
General Form

• This example demonstrates the simplest general form of an Awk program:

• `awk <search pattern> {<program actions>}`
• `<search pattern>` is a test
• `{<program actions>}` is the action to perform if the test is passed.
If statement

• I want to list all the coins that were minted before 1980. I invoke Awk as follows:

  awk '{if ($3 < 1980) print $3, " ",$5,$6,$7,$8}'
coins.txt

• This yields:

• 1908 Franz Josef 100 Korona
• 1979 Krugerrand
The next example prints out how many coins are in the collection:

```
awk 'END {print NR,"coins"}' coins.txt
```

This yields:

```
4 coins
```
general form of an Awk program

• the general form of an Awk program to:
• awk 'BEGIN {<initializations>}
• <search pattern 1> {<program actions>}
• <search pattern 2> {<program actions>} ... END {<final actions>}'
Example to calculate total gold

- Suppose the current price of gold is $425, and I want to figure out the approximate total value of the gold pieces in the coin collection. I invoke Awk as follows:

  ```awk '/gold/ {ounces += $2} END {print "value = $" 425*ounces}' coins.txt```

- This yields: (note ounces is user defined)
  - value = $2592.5
Step by step

• So the program action:
• \{ounces += $2\}
• Another way of saying:
• \{ounces = ounces + $2\}
• The final action is to compute and print the value of the gold:
• END {print "value = $" 425*ounces}
AWK PROGRAM EXAMPLE

• Instead of doing it all from the command line
• We can do it all from a file,
• With the following syntax
• *awk* -f <*awk program file name*> (*awk* is another example of a scripting language)
Output of follow program

• Summary Data for Coin Collection:
• Gold pieces: nn
• Weight of gold pieces: nn.nn
• Value of gold pieces: n,nnn.nn
• Silver pieces: nn
• Weight of silver pieces: nn.nn
• Value of silver pieces: n,nnn.nn
• Total number of pieces: nn
• Value of collection: n,nnn.nn
# This is an awk program that summarizes a gold coin collection.

- `/gold/ { num_gold++; wt_gold += $2 }` # Get weight of gold.
- `END { val_gold = 485 * wt_gold; }` # Compute value of gold.
- `print "Summary data for coin collection:";` # Print results.
- `printf ("\n");`
- `printf (" Gold pieces: %2d\n", num_gold);`
- `printf (" Weight of gold pieces: %5.2f\n", wt_gold);`
- `printf (" Value of gold pieces: %7.2f\n", val_gold); printf ("\n");`
- `printf ("\n"); printf (" Total number of pieces: %2d\n", NR);`
- `printf (" Value of collection: %7.2f\n", total);`
tutorial

- [http://www.vectorsite.net/tsawk_1.html#m1](http://www.vectorsite.net/tsawk_1.html#m1)
- You can cut and paste the commands from here or from these slides.
- You can easily look at this in your 2 hours private study time