Session 1

Regular Expressions

Yong Zhang, Ph D

Mar 7, 2019

Outline

- Introduction to UNIX shell
- Regular expressions
  - Introduction to regular expressions
  - **grep** command
- Basic Unix commands

What is Unix

- Powerful operating system developed at the Bell Labs
- Popular in scientific, engineering and academic communities
- Multi-user and multi-tasking
- Runs on different computer hardware

Brief History of Unix

- Multics (1964-1969)
  - AT&T Bell Lab, MIT, GE
- Unix (since 1971)
  - Ken Thompson & Dennis Ritchie
- BSD (since 1978)

Unix Philosophy

- Small is beautiful.
- Make each program do one thing well.
- Build a prototype as soon as possible.
- Choose portability over efficiency.
- Store data in flat text files.
- Use software leverage to your advantage.
- Use shell scripts to increase leverage and portability.
- Avoid captive user interfaces.
- Make every program a filter.

--- Mike Gancarz (1994)

What is Linux

- The Linux **kernel** was developed by **Linus Torvalds** in early 1990's and he made the code free
- Kernel: http://en.wikipedia.org/wiki/Kernel_(computing)
- Others quickly extended the kernel and developed Unix-like programs
- All the source code is free to study, redistribute, and modify

GNU / Linux

Richard M Stallman

Linus B Torvalds

3/7/19

Command Line Shell

– Unix has many command line programs

– Issued via a **shell** (command line interface)
- Shell provides structure and a programmable interface. Two popular ones:
  - **bash** the Bourne-again shell (standard on Linux)
  - **tcsh** based on csh or C shell
- Check which shell is current running: echo $SHELL

3/7/19

Command Examples

– Recall previous commands using the **UP** key
– Where am I: pwd
– Change directory: cd .. cd ~

3/7/19

Online Documentation

– man command: show full manual document
- man ls
- man man
- **Space** key to go down a page, **Enter** to go down a line, "**q**" to quit

– whatis command: one-line summary of a command
- whatis ls

– apropos command: search by keyword
- apropos login

3/7/19

Edit and Examine Files

– Use the pico editor to edit a file.
– Examine a simple text file: cat foo1
– Examine a long file:
- more foo1
- less foo1

– Examine part of a file
- first n lines: head -n foo1
- last lines starting at n: tail -n foo1

– Determine the type of a file: file foo1

3/7/19

Foreground and Background Jobs

– Processes can run in two modes:
- **foreground** (no prompt back unit it finishes)
- **background** (you can go on to other tasks while they execute), also called **batch jobs**.

– A foreground job
- receives keyboard input and signals such as Control-C
- is terminated if log out

– A background job continues to run even log out

3/7/19

Run a Job in Background

1. Initiate a job in the background: &

1. Suspend a foreground job with control-Z, and then let it run in the background with the **bg** command:
   Control-Z
   bg

Job Control Commands

- Identify jobs in the **current session**: jobs or jobs -l
- List the current running processes status: ps -l
- Suspend the foreground job: key Control-Z
- Kill the foreground job: Control-C

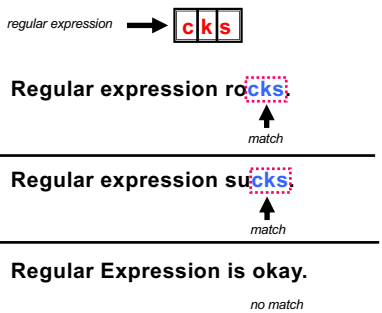
Outline

- Introduction to UNIX shell
- Regular expressions
  - Introduction to regular expressions
  - **grep** command
- Basic Unix commands

What is a Regular Expression?

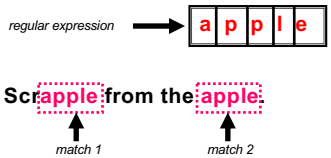
- A regular expression (*regex*) describes a pattern to match multiple input strings.
- Regular expressions are endemic to Unix
  - Some utilities/programs that use them:
    - **vi**, **ed**, **sed**, and **emacs**
    - **awk**, **tcl**, **perl** and **python**
    - **grep**, **egrep**, **fgrep**
    - **compilers**
- The simplest regular expression is a string of literal characters to match.
- The string matches the regular expression if it contains the substring.

Regular Expressions: Exact Matches



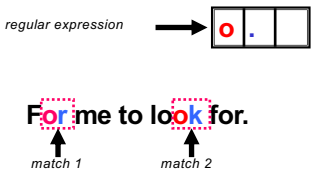
Regular Expressions: Multiple Matches

- A regular expression can match a string in more than one place.



Regular Expressions: Matching Any Character

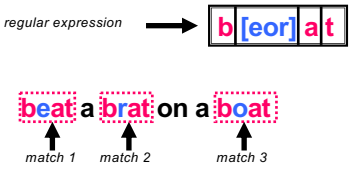
– The . regular expression can be used to match any character.



3/7/19

Regular Expressions: Alternate Character Classes

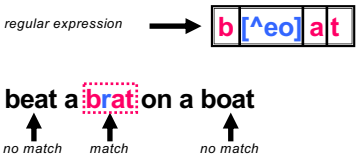
– Character classes [] can be used to match any specific set of characters.



3/7/19

Regular Expressions: Negated Character Classes

– Character classes can be negated with the [^] syntax.



3/7/19

Regular Expressions: Other Character Classes

– Other examples of character classes:
  – [aeiou] will match any of the characters a, e, i, o, or u
  – [kK]orn will match korn or Korn
– Ranges can also be specified in character classes
  – [1-9] is the same as [123456789]
  – [abcde] is equivalent to [a-e]
– You can also combine multiple ranges
  – [abcde123456789] is equivalent to [a-e1-9]
– Note that the - character has a special meaning in a character class but only if it is used within a range
  – [-123] would match the characters -, 1, 2, or 3

3/7/19

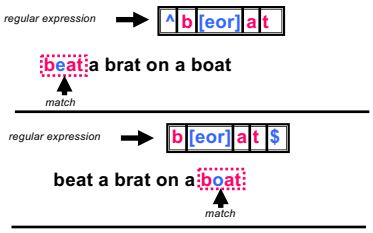
Regular Expressions: Named Character Classes

– Commonly used character classes can be referred to by name
  – alpha, Any alpha character A to Z or a to z
  – lower, Any alpha character a to z
  – upper, Any alpha character A to Z
  – alnum, Any alphanumeric character 0 to 9 or A to Z or a to z
  – digit, digits 0 to 9
  – punct, Punctuation symbols . , " ' ? ! ; : # $ % & ( ) * + - / < > = @ [ ] \ ^ _ { } | ~
  – space, Any whitespace characters space, tab, NL, FF, VT, CR
– Syntax [:name:]
  – [a-zA-Z] ⇔ [[:alpha:]]
  – [a-zA-Z0-9] ⇔ [[:alnum:]]
  – [45a-z] ⇔ [45[:lower:]]

3/7/19

Regular Expressions: Anchors

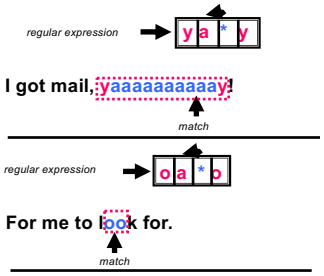
– Anchors are used to match at the beginning or end of a line
  – ^ means beginning of the line
  – $ means end of the line



3/7/19

Regular Expression: Repetitions

- The * is used to define **zero or more** occurrences of the *single* regular expression preceding it.



3/7/19

Regular Expressions: Repetition Ranges, Subexpressions

- Ranges can also be specified for repetition
  - `\{n,m\}` notation can specify a range of repetitions for the immediately preceding regex
    - `\{n\}` means exactly *n* occurrences
    - `\{n,\}` means at least *n* occurrences
    - `\{n,m\}` means at least *n* occurrences but no more than *m* occurrences
  - Example:
    - `.\{0,\}` same as `.*`
    - `a\{2,\}` same as `aaa*`
- If you want to group part of an expression so that * applies to more than just the previous character, use `\( \)` notation
  - Subexpresssions are treated like a single character
  - `\(abc\)\{2,3\}` matches `abcabc` or `abcabcabc`

3/7/19

Single Quoting Regex

- Since many of the special characters used in regexs also have special meaning to the shell, it's a good idea to get in the habit of single quoting your regexs.
  - This will protect any special characters from being operated on by the shell
  - If you habitually do it, you won't have to worry about when it is necessary
- Even though we are single quoting our regexs so the shell won't interpret the special characters, sometimes we still want to use an operator as itself. To do this, we escape the character with a \ (backslash)
  - Suppose we want to search for the character sequence 'a*b*'
    - Unless we do something special, this will match zero or more 'a's followed by zero or more 'b's, **not what we want!**
    - 'a*b*' will fix this - now the asterisks are treated as regular characters

3/7/19

Regular Expressions: Backreferences

- Sometimes it is handy to be able to refer to a match that was made earlier in a regex
- This is done using backreferences
  - \n is the backreference specifier, where n is a number
- For example
  - `^\([[:alpha:]]\{1,\}\).*\1$`
  - `\(["']\).*\1`

3/7/19

Extended Regular Expressions

- Regex also provides an alternation character | for matching one or another subexpression
  - `(T|Fl)an` will match Tan or Flan
  - `^(From|Subject):` will match the From and Subject lines of a typical email message
    - It matches a beginning of line followed by either the characters From or Subject followed by a ':'
- Subexpressions are used to limit the scope of the alternation
  - `At(ten|nine)tion` then matches Attention or Atninetion, not Atten or ninetion as would happen without the parenthesis - `Atten|ninetion`

3/7/19

Extended Regular Expressions: Repetition Shorthands

- The * (star) has already been seen to specify zero or more occurrences of the immediately preceding character
- The + (plus) means one or more
  - abc+d will match abcd, abccd, or abccccccd but will not match 'abd' while abc?d will match abd and abcd but not 'abccd'
  - Equivalent to {1,}
- The ? (question mark) specifies an optional character, the single character that immediately precedes it
  - `July?` will match Jul or July
  - Equivalent to `{0,1}`; also equivalent to `(Jul|July)`
- The *, ?, and + are known as quantifiers because they specify the quantity of a match

3/7/19

Regular Expressions: Some Practical Examples

- Variable names in C
- Dollar amount with optional cents
- Time of day
- HTML headers <h1> <H1> <h2> …

Regex Metacharacters

\b Matches a word boundary, that is, the position between a word and a space. For example, er\b matches the er in "never" but not the er in verb.
\B Matches a nonword boundary. ea*r\B matches the ear in never early.
\d Matches a digit character. Equivalent to [0-9].
\D Matches a nondigit character. Equivalent to [^0-9].
\n Matches a newline character.
\r Matches a carriage return character.
\s Matches any white space including space, tab, form-feed, etc. Equivalent to [\f\n\r\t\v].
\S Matches any nonwhite space character. Equivalent to [^ \f\n\r\t\v].
\t Matches a tab character.
\v Matches a vertical tab character.
\w Matches any word character including underscore. Equivalent to [A-Za-z0-9_].
\W Matches any nonword character. Equivalent to [^A-Za-z0-9_].

Outline

- Introduction to UNIX shell
- Regular expressions
  - Introduction to regular expressions
  - **grep** command
- Basic Unix commands

grep command

- grep comes from the **ed** (Unix text editor) search command "**g**lobal **r**egular **e**xpression **p**rint"
- This was such a useful command that it was written as a standalone utility
- There are two other variants, ***egrep*** and ***fgrep*** that comprise the *grep* family
- *grep* is the answer to the moments where you know you want the file that contains a specific phrase but you can't remember its name

Family Differences

- **grep** - uses regular expressions for pattern matching
- **fgrep** - file grep, does not use regular expressions, only matches fixed strings but can get search strings from a file
- **egrep** - extended grep, uses a more powerful set of regular expressions but does not support backreferencing, generally the fastest member of the grep family
- grep and egrep have different syntax
  - **grep**: BREs
  - **egrep**: EREs

Regular Experessions: Quick Refrences

x xyz	Ordinary characters match themselves (NEWLINES and metacharacters excluded) Ordinary strings match themselves	fgrep, grep, egrep
\m ^ $. [xy^$x] [^xy^$z] [a-z] r* r1r2	Matches literal character *m* Start of line End of line Any single character Any of x, y, ^, $, or z Any one character other than x, y, ^, $, or z Any single character in given range zero or more occurrences of regex r Matches r1 followed by r2	grep, egrep
\(r\) \n \{n,m\}	Tagged regular expression, matches r Set to what matched the *n*th tagged expression (n = 1-9) Repetition	grep
r+ r? r1|r2 (r1|r2)r3 (r1|r2)* {n,m}	One or more occurrences of r Zero or one occurrences of r Either r1 or r2 Either r1r3 or r2r3 Zero or more occurrences of r1|r2, e.g., r1, r1r1, r2r1, r1r1r2r1,…) Repetition	egrep

grep Usage

- *grep -[options] 'pattern' filenames*
  - **-c** Prints only a count of matching lines, rather than printing the matching lines themselved
  - **-i** Ignores uppercase/lowercase distinctions in both file and pattern
  - **-n** Print lines and line numbers for each occurrence of pattern match
  - **-l** Prints filenames containing matches to pattern, but not matching lines
  - **-h** Prints matching lines but not filenames (opposite to -l)
  - **-v** Prints only those lines that don't contain a match with pattern

Exercise

- Use "grep" and "wc" to count the total number of DNA sequences in yeast_gene.fa (FASTA format).
- Use "grep" and "wc" to count the total number of nucleotides in yeast_gene.fa

Outline

- Introduction to UNIX shell
- Regular expressions
  - Introduction to regular expressions
  - **grep** command
- Basic Unix commands

List Contents and Directories

- ls command
- list in **l**ong format: ls -l
- list only name of a **d**irectory: ls -d
- list entries sorted by **t**ime: ls -t
- **r**everse the order of the sort: ls -r
- recursively list content of any directories: ls -R
- list hidden files: ls -a
- Can combine multiple options: ls -lrt

Create Directories

- mkdir to make a new directory:
- mkdir dir1
- rm to remove a directory and any files it contains (use with caution) :
- rm -r dir1
- mv dir1 dir2
- If dir2 does not exist, renames dir1 to dir2
- If dir2 exists, moves dir1 inside dir2
- cp -r dir1 dir2 ('r' means recursive)
- If dir2 does not exist, copies dir1 as dir2
- If dir2 exists, copies dir1 inside dir2

Create and Copy Files

- touch foo1: Create an empty file foo1
- mv foo1 foo2: Rename foo1 to foo2 (overwrites original contents of foo2, if it exists)
- cp foo2 foo3: Copy foo2 as foo3 (overwrites original contents of foo3, if it exists)
- rm foo3: Remove foo3
- -i: requesting confirmation for each removal: rm -i foo2
- cp foo1 dir1: Copy file foo1 into existing directory dir1
- mv foo2 dir2: Move file foo2 into existing directory dir2

File Permission

- There are three user classes:
  - owner/user (u), group (g), others (o)
- There are three types of file permission modes:
  - read (r), write (w), execute (x)
- Look at permissions: ls -l foo1
  - All 3 permission modes for 3 user groups are ordered ("d" for directory)
  - see group memberships: groups username

Change File Permissions

- Change file permission modes: chmod
- Use class and action abbreviations
  - chmod u+x,g+rw,o=rw foo1
  - chmod o+r *
- Use numeric arguments
  - for every user class, the 3 mode values are regarded as a binary number
  - chmod 751 foo1
- Change permissions recursively for all files in the directory
  - chmod -R o+rx dir1

Redirect Output/Input

- Output: >, >>
  - redirect: ls > foo2
  - prevent overwriting of files: set -o noclobber
  - append to the end of an existing file:
    - ls >> foo2
- Input: <
- Pipes: |
  - who | sort (sort the current users found by who)
  - who>foo3
  - sort<foo3>foo4

Unix Wildcards

- ? represents a single character:
- * represents any number of characters (including none)
- A range of characters: [ ]
  - ls foo[12]
  - ls foo[1-4]
  - ls [fm]*

Renaming Commands with Alias

- Create a temporary aliases: alias ll='ls -l'
- Check aliases: alias
- Remove an alias: unalias ll

Job Control Commands

- fg or fg %n: Bring the most recently backgrounded job (or specific job) to the foreground:
- bg or bg %n: Run the most recently stopped job (or specific job) in the background
- Kill job: kill %n (n is job number)
- Kill process: kill -9 1234 (1234 is PID)

tr: Translate Characters

- Copies standard input to standard output with substitution or deletion of selected characters
- Syntax: *tr [-cds] [string1] [string2]*
  - `-d` delete all input characters contained in *string1*
  - `-c` complements the characters in *string1* with respect to the entire ASCII character set
  - `-s` squeeze all strings of repeated output characters in the last operand to single characters

tr (continued)

- *tr* reads from standard input.
  - Any character that does not match a character in *string1* is passed to *standard output* unchanged
  - Any character that does match a character in *string1* is translated into the corresponding character in *string2* and then passed to *standard output*
- Examples
  - *tr s z* replaces all instances of *s* with z
  - *tr so zx* replaces all instances of *s* with *z* and *o* with *x*
  - *tr a-z A-Z* replaces all lower case characters with upper case characters
  - *tr -d a-c* deletes all a-c characters

tr Examples

- Change delimiter
- Rewrite numbers
- Remove all two more successive blank spaces, and replace the blank space to tab
- Find printable ASCII in a binary file

wc: Counting results

- The word count utility, **wc**, counts the number of lines, characters or words
- Options:
  - `-l` Count lines
  - `-w` Count words
  - `-c` Count characters
- Default: count lines, words and chars

cat: The simplest filter

- The cat command copies its input to output unchanged (*identity filter*). When supplied a list of file names, it con**cat**enates them onto stdout.
- Some options:
  - `-n` **n**umber output lines (starting from 1)
  - `-v` display control-characters in **v**isible form (e.g. ^C)
- Examples
  - ***cat file1***
  - ***cat file1 file2 > file3***

Unix Text Files: Delimited Data

Tab Separated

```
John     99
Anne     75
Andrew   50
Tim      95
Arun     33
Sowmya   76
```

Pipe-separated

```
COMP1011|2252424|Abbot, Andrew John |3727|1|M
COMP2011|2211222|Abdurjh, Saeed |3640|2|M
COMP1011|2250631|Accent, Aac-Ek-Murhg |3640|1|M
COMP1021|2250127|Addison, Blair |3971|1|F
COMP4012|2190705|Allen, David Peter |3645|4|M
COMP4910|2190705|Allen, David Pater |3645|4|M
```

Colon-separated

```
root:ZHolHAHZw8As2:0:0:root:/root:/bin/ksh
jas:nJz3ru5a/44Ko:100:100:John Shepherd:/home/jas:/bin/ksh
cs1021:iZ3sO9OO5eZY6:101:101:COMP1021:/home/cs1021:/bin/bash
cs2041:rX9KwSSPqkLyA:102:102:COMP2041:/home/cs2041:/bin/csh
cs3311:mLRiCIvmtI9O2:103:103:COMP3311:/home/cs3311:/bin/sh
```

cut: Select Columns

- The cut command prints selected parts of input lines.
  - can select columns (assumes tab-separated input)
  - can select a range of character positions
- Some options:
  - **-f** *listOfCols*: print only the specified columns (tab-separated) on output
  - **-c** *listOfPos*: print only chars in the specified positions
  - **-d** *c*: use character *c* as the column separator
- Lists are specified as ranges (e.g. 1-5) or comma-separated (e.g. 2,4,5).

3/7/19

cut Examples

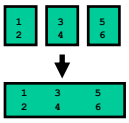
```
cut -f 1 < data
cut -f 1-3 < data
cut -f 1,4 < data
cut -f 4- < data
cut -d '|' -f 1-3 < data
cut -c 1-4 < data
```

Unfortunately, there's no way to refer to "last column" without counting the columns.

3/7/19

paste: Join Columns

- The paste command displays several text files "in parallel" on output.
- If the inputs are files **a**, **b**, **c**
  - the first line of output is composed of the first lines of **a**, **b**, **c**
  - the second line of output is composed of the second lines of **a**, **b**, **c**



- Lines from each file are separated by a tab character.
- If files are different lengths, output has all lines from longest file, with empty strings for missing lines.

3/7/19

paste Example

```
cut -f 1 < data > data1
cut -f 2 < data > data2
cut -f 3 < data > data3
paste data1 data3 data2 > newdata
```

3/7/19

sort: Sort lines of a file

- The sort command copies input to output but ensures that the output is arranged in ascending order of lines.
  - By default, sorting is based on ASCII comparisons of the whole line.
- Other features of sort:
  - understands text data that occurs in columns. (can also sort on a column other than the first)
  - can distinguish numbers and sort appropriately
  - can sort files "in place" as well as behaving like a filter
  - capable of sorting *very large* files

3/7/19

sort Options

- Syntax: *sort [-dftnrk] [-o filename] [filename(s)]*
  - *-d* Dictionary order, only letters, digits, and whitespace are significant in determining sort order
  - *-f* Ignore case (fold into lower case)
  - *-t* Specify delimiter
  - *-n* Numeric order, sort by arithmetic value instead of first digit
  - *-r* Sort in reverse order
  - *-k* Key
  - *-o filename* write output to filename, filename can be the same as one of the input files

3/7/19

sort Examples

- `sort -t: -k1,1 /etc/passwd`
- `sort -t: -k3nr,3 /etc/passwd`

```
chico:x:12501:1000:Chico Marx:/home/chico:/bin/bash
daemon:x:2:2:daemon:/sbin:/sbin/nologin
groucho:x:12503:2000:Groucho Marx:/home/groucho:/bin/sh
gummo:x:12504:3000:Gummo Marx:/home/gummo:/usr/local/bin/ksh93
harpo:x:12502:1000:Harpo Marx:/home/harpo:/bin/ksh
root:x:0:0:root:/root:/bin/bash
zeppo:x:12505:1000:Zeppo Marx:/home/zeppo:/bin/zsh
```

- `sort -o mydata mydata`

uniq: List unique items

- Remove or report adjacent duplicate lines
- Syntax: *uniq [-cdu] [input-file] [output-file]*
  - **`-c`** Supersede the -u and -d options and generate an output report with each line preceded by an occurrence count
  - **`-d`** Write only the duplicated lines
  - **`-u`** Write only those lines which are not duplicated
  - The default output is the union (combination) of *-d* and *–u*
- *Examples*
  - **`sort file | uniq | wc –l`**