

Pattern Matching and Regular Expressions
sed & awk

Intro to Unix Spring 2000 Pattern Matching 1

egrep sed vi awk perl

- All these Unix commands support using *regular expressions* to describe patterns.
- There are some minor differences between the regular expressions supported by these programs – see the book for details.
- We will cover the general matching operator.

Intro to Unix Spring 2000 Pattern Matching 2

Not Filename Expansion!

- Although there are similarities to the metacharacters used in filename expansion – we are talking about something different!

Filename expansion is done by the shell.

Regular expressions are used by commands (programs).

Intro to Unix Spring 2000 Pattern Matching 3

Search Patterns

- Any character (except a metacharacter!) matches itself.
- The "." character matches any character except newline.

"**F.**" Matches an 'F' followed by any character.

"**a.b**" Matches 'a' followed by any 1 char followed by 'b'.

Intro to Unix Spring 2000 Pattern Matching

4

Searching for Metacharacters

If you really want to match '.', you can use "\."

Regex	Matches	Does not match
a.b	axb	abc
a\.b	a.b	axb

Intro to Unix Spring 2000 Pattern Matching

5

Character Class

[**abc**] matches a single **a b** or **c**

[**a-z**] matches any of **abcdef...xyz**

[**^A-Za-z**] matches a single character as long as it is not a letter.

Intro to Unix Spring 2000 Pattern Matching

6

[Dd][Aa][Vv][Ee]

- Matches "Dave" or "dave" or "dAVE",
- Does not match "ave" or "da"

Repetition using *

* means 0 or more of the previous single character pattern.

[abc]* matches "aaaa" or "acbca"

Hi Dave.* matches "Hi Dave" or
"Hi Daveisgoofy"

0*10 matches "010" or "0000010" or "10"

Repetition using +

+ means 1 or more of the previous single character pattern.

[abc]+ matches "aaaa" or "acbca"

Hi Dave.+ matches "Hi Dave." or
"Hi Dave...."

0+10 matches "010" or "0000010"
does not match "10"

? Repetition Operator

? means 0 or 1 of the previous single character pattern.

x[abc]?x matches "xax" or "xx"

A[0-9]?B matches "A1B" or "AB"
does not match "a1b" or "A123B"

Using regexps with grep

grep regexp files...

egrep [a-z][0-9] file1 NO!

egrep "[a-z][0-9]" file1 YES!

Grouping with parens

- If you put a subpattern inside parens you can use + * and ? to the entire subpattern.

a(bc)*d matches "ad" and "abcbcd"
does not match "abcxd" or "bcbcd"

Grouping and Memory

- The string that matches the stuff in parentheses can be used later in the regular expression:

```
([a-z]+)[ \t]+\1
```

matches "n n" or "book book"

More than one memory

- `\1` is the substring that matches the first regexp in parenthesis.
- `\2` matches the second substring ...
- Up to `\9`

What does this match?

```
http://([a-z])(\1)\2\.*\.com
```

Sed – stream editor

- Supports many types of editing operations, we look at one very useful one – substitution.

s/regexp/replacement/modifier

```
sed 's/[aeiou]/\$/g'
```

sed Command Line

```
sed [-n] -f scriptfile file(s)
      -or-
sed [-n] [-e] 'command' files(s)
```

- n means suppress default output (doesn't print anything unless you tell it to).

-f commands are in the file **scriptfile**

-e used when specifying multiple commands from the command line.

sed Commands

```
[address[, address]][! ]command[arguments]
```

- Each command is applied to any input lines that *match* the address(es).
- The commands are editing commands:
 - append, replace, insert, delete, substitute, translate, print, copy, paste,...

sed addresses

- The address for a command can be:
 - No address: matches every line of input
 - 1 address: any line that matches the address.
 - 2 addresses: lines that are between a line that matches the first address and one that matches the second address (inclusive).
 - Address followed by !: any line that would not be matched by the address.

sed addresses (cont.)

- Each address can be:
 - line number
 - the '\$' character (means the previous line).
 - a regular expression inside /s
`/foo/`
`/[hH][eE][aA][dD]/`

Some Commands

- d** delete line
- p** print line
- h** copy to temp buffer
- g** paste (replace with temp buffer).

Some examples

`/Windows/d` deletes lines that contain the word Windows.

`/[Uu]nix/d` deletes lines that do not contain the word unix.

`/[0-9]/p` prints lines that contain any digits.

Entire command lines

> `sed '1,5/d' somefile`
prints everything but the first 5 lines of the file somefile
(same as `tail +6 somefile`).

> `sed -n '/foo/p' /usr/dict/words`
just like `grep foo /usr/dict/words`

> `sed -n '/START/,/STOP/p'`
prints everything from STDIN between
a line that contains START and one the contains STOP

substitutions

`s/regexp/replacement/[flags]`

– replaces anything that matches the regular expression with the replacement text.

`sed -n 's/[0-9]/#/'`
matches every line (no addresses!)
replaces the first digit on line with “#”

Substitution Flags

- g** global (replace all matches on the line).
- p** print the line if a successful match
- w** write the line to a file if a match was found.
- n** replace the nth instance (match) only.

Intro to Unix Spring 2000 Pattern Matching

25

substitution examples

```
sed 's/[wW]indows/Unix/g'
```

```
sed -n 's/f/F/gp'
```

print every line containing 'f' after
replacing all 'f's by 'F's.

Intro to Unix Spring 2000 Pattern Matching

26

sed scripts

- Series of commands can be put in a file and use the '-f' option.
- Can also create an sed script:

```
#!/bin/sed -nf  
s/vi/emacs/g  
/[Ww]indows/d  
p
```

Intro to Unix Spring 2000 Pattern Matching

27

Possibly Interesting Example

- sed script to remove all HTML tags from a file:

```
#!/bin/sed -nf
# sed script to remove tags

s/<[^>]*>/ /g
P
```

Intro to Unix Spring 2000 Pattern Matching

28

Even more interesting

- Replace all HTML tags with the tag name (something like would become **B**)

```
#!/bin/sed -nf
#
s/<[ \t]\|/?*([^\>]*)>/\1/g
P
```

whatever matched the regexp inside the parens!

Intro to Unix Spring 2000 Pattern Matching

29

awk

- Pattern matching program – useful for automating complex text-handling chores.
- You create a *script* that is a series of patterns and corresponding actions (sounds like **sed**, but is really quite different).

Intro to Unix Spring 2000 Pattern Matching

30

Awk script format

```
/pattern1/ { actions... }
```

```
/pattern2/ { actions... }
```

```
BEGIN { actions... }
```

← these actions happen before input is read!

```
END { actions... }
```

← these actions happen after all the input is read!

Intro to Unix Spring 2000 Pattern Matching

31

Awk and input

- Awk automatically splits each line of input on the field separator FS (by default whitespace).
- Awk creates the variables \$1, \$2, \$3... that correspond to the resulting fields (just like a shell script). \$0 is the entire line.

Intro to Unix Spring 2000 Pattern Matching

32

Lots of built-in variables

FILENAME the current input file

FS input field separator

NF number of fields in the current record

NR number of the current record (line)

Intro to Unix Spring 2000 Pattern Matching

33

Example

```
awk '{
for (i=1; i <= NF; i++)
  printf("WORD %d : %s\n",i,$i)
}'
```

Intro to Unix Spring 2000 Pattern Matching

34

Awk ing the passwd file

```
awk -F: '{
for (i=1; i <= NF; i++)
  printf("WORD %d : %s\n",i,$i)
}' passwd
```

Intro to Unix Spring 2000 Pattern Matching

35

Awk script that adds lline numbers to a text file.

```
#!/bin/awk -f
BEGIN {
printf("Line numbers by awk\n");
}
{
printf("%d: %s\n",NR,$0);
}
```

Intro to Unix Spring 2000 Pattern Matching

36

Other stuff in AWK

- Lots of control structures (if, while, for,...)
- library of functions (subroutines)
- You can write your own functions*
- Ability to run external programs.

* This is only supported by nawk

ls filtered by awk (HW1)

```
#!/bin/awk -f
BEGIN {
  while ( `ls -ald *` | getline ) {
    print $5,$9 | "sort -rn";
  }
}
```
