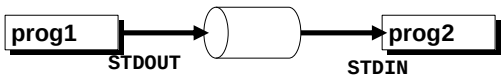
Misc. Topics: Pipes, Regular Expressions, Makefiles

Pipes

- A pipe is a holder for a stream of data.
- A pipe can be used to hold the output of one program and feed it to the input of another.



Asking for a pipe

- Separate 2 commands with the "|" character.
- The shell does all the work!

```
ls | sort
ls | sort > sortedls
```

- Building commands: You can string together a series of unix commands to do something new!

Regular Expressions

`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 documentation for details.
- We will cover the general matching operator.

Not Filename Expansion!

- Although there are similarities to the metacharacters used in filename expansion (called globs) – we are talking about something different!
- Filename expansion is done by the shell.
- Regular expressions are used by commands (programs).

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'.

Searching for Metacharacters

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

Regexp	Matches	Does not match
`a.b`	`axb`	`abc`
`a\.b`	`a.b`	`axb`

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.

`[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 "aaaaa" 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 "aaaaa" 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

- If you want to edit a single file to do some global operation such as find-replace, insert, append, delete,… you'll just open it in vi or ms-word and edit it.
- If you want to edit *several* files to do such global operations, you'll use sed.
- E.g. Removing HTML tags, changing author names ☺, changing variable names in source code
- Supports many types of editing operations, we look at one very useful one – substitution.

`s/regexp/replacement/modifier`

`cat file | sed 's/[aeiou]/\$/g'`

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,…

awk

- If you want to do spread-sheet like computations, you'll use 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 – supports more complex actions)
- You can declare variables, do math, etc.

Awk script format

```
/pattern1/ { actions… }

/pattern2/ { actions… }

BEGIN { actions… }

END { actions… }
```

these actions happen before input is read!

these actions happen after all the input is read!

Example

- Adding column 2 in a file that contains columns of numbers:

```
cat file | awk '{n += $2} END {print n}'
```


Makefiles

- The make command automates the process of building files that *depend* on other files.
- Typically used for program development:
  - runs the compiler only when necessary.
  - uses file access times to decide when it is necessary.
- make can be used for lots of tasks! (not just for programming).

Dependency

- File **`foo`** should be *rebuilt* whenever file **`blah`** is changed.
  - if **`blah`** is newer than **`foo`**, we need to rebuild **`foo`**.
  - **`foo`** *depends* on **`blah`**

Simple Makefile for building a program.

TABS!

```
subs.o: subs.c
        gcc –c subs.c

main.o: main.c
        gcc –c main.c

main: main.o subs.o
        gcc –o main main.o subs.o
```


Advanced Makefile for building a program.

```
.c.o:           # suffix rule
        gcc –c $<           # $< - prereq

SOURCE = main.c subs.c #macro
OBJS = ${SOURCE:.c=.o}
                #macro substitution

main: $(OBJS)
        gcc –o $@ $(OBJS)     # $@ - target
```


The End