Elaborazione dati dalla riga di comando

Luca Ceresoli

Elaborare dati

Sistemi per elaborare dati

Informatica = elaborazione automatica delle informazioni

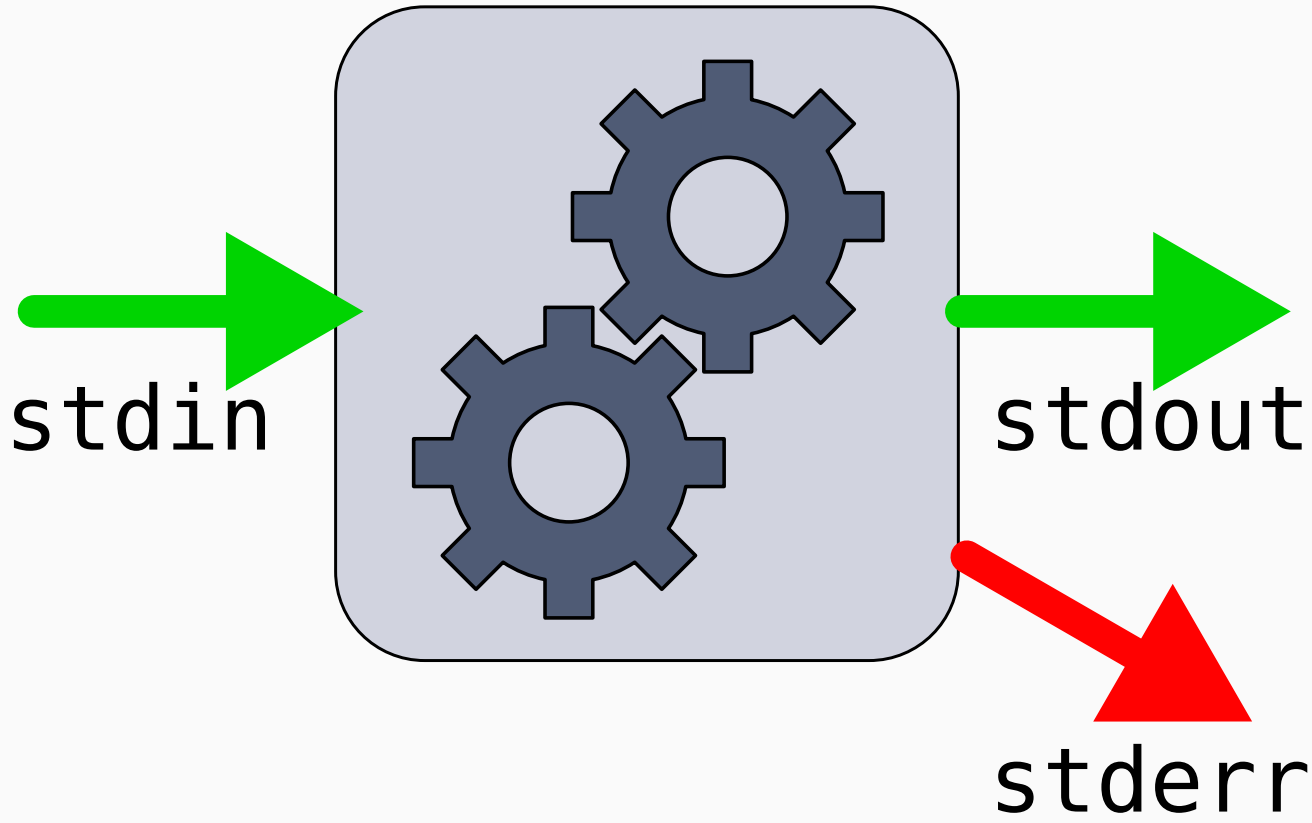
- Database relazionali
- Spreadsheet
- Software ad-hoc
- ...
- La riga di comando

La riga di comando

La riga di comando

- Strumento classico, nato ai tempi dai "terminali"
- Prompt: richiede un "comando", lo esegue, torna al prompt
- Esistono migliaia di possibili comandi
- Molto spartana, molto potente
  - Presente su praticamente ogni sistema UNIX-like
  - Utilizzabile su sistemi poco carrozzati...
  - ...o remoti, con connessioni lentissime

Standard input, standard output, standard error



Visualizzare i dati

- `cat`
- `less`
- `wc`
- `head`, `tail`
- Demo!

Semplici elaborazioni

- `sort`
- `cut`
- Demo!

grep

grep

- Filtra le righe, lasciando passare solo le linee che corrispondono a un criterio (pattern)
- Utilizzo base:
  - `grep` *`PATTERN`* `file1 file2 file3`
  - `grep` *`PATTERN`*
    - Filtra lo standard input
- Demo!

grep: varianti

- Tipi di pattern:
  - `grep -F` *`PATTERN`* — Ricerca esatta
  - `grep -G` *`PATTERN`* — Basic regular expressions
  - `grep -E` *`PATTERN`* — Extended regular expressions

grep: opzioni aggiuntive

- `-i`: case insentitive
- `-w`: solo parole intere
- `-v`: invert selection: mostra righe *non* corrispondenti
- `-l`: mostra solo i nomi dei files corrispondenti
- Demo!

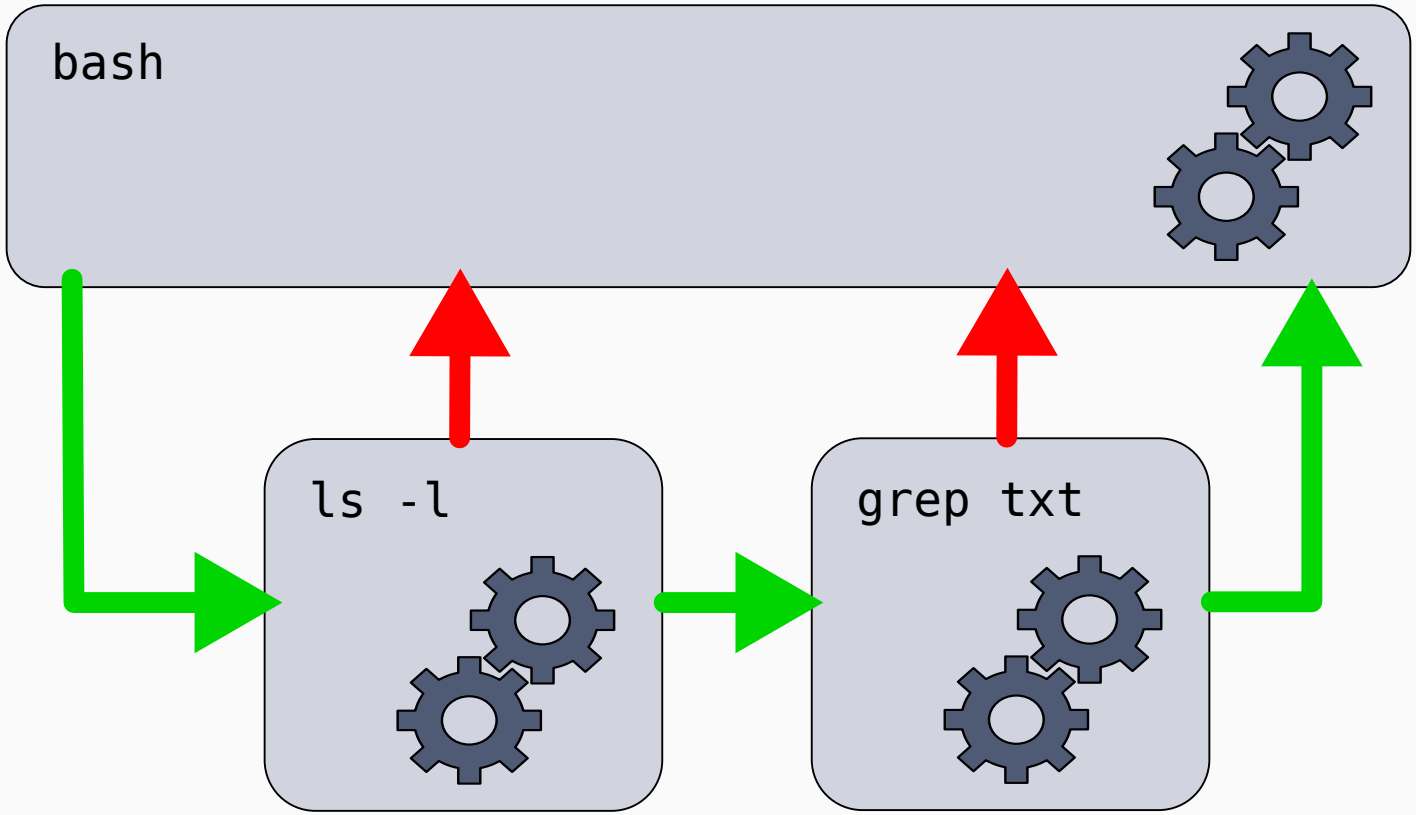
La pipeline

La pipeline

- Simbolo: |
- Permette di collegare comandi "in catena"
- Principio UNIX: ogni software svolge una funzione
  - La svolge in modo flessibile ed efficiente
  - Si integra con altri strumenti che svolgono altre funzioni

La pipeline

Esempio: `ls -l | grep txt`



Semplici elaborazioni con la pipeline

- `uniq`
- `tr`
- Demo!

Espressioni regolari

Espressioni regolari

Espressioni regolari = Regular expressions = regexp

- Un linguaggio per descrivere un pattern
- Elabora una stringa (= sequenza di caratteri)
  - Esito: match o non match
- Usato da grep, sed e vari altri strumenti

Sottostringa

- Sottostringa: regexp senza caratteri speciali
  - Esempio: “in”
  - Significato: tutte le stringhe contenenti “in”
  - Match: “**in**”, “L**in**ux”, “P**in**guino”
  - No match: “MacOS”, “UNIX”

Qualsiasi carattere

- Punto: qualsiasi carattere
  - Esempio: ".in.x"
  - Significato: tutte le stringhe contenenti un carattere qualsiasi, poi "in", poi un carattere qualsiasi, poi "x"
  - Match: "**Linux**", "**Minix**", "GNU **Linux**"
  - No match: "inox", "Linoox"

Un carattere da un elenco

- `[`*caratteri*`]`: un carattere da un elenco
  - Esempio: "`c[ao]sa`"
  - Match: "**casa**", "**cosa**"
  - No match: "cisa"
- `[`*x*`-`*y*`]`: un range di caratteri
  - Esempio: "`casa[a-z]`"
  - Match: "rin**casar**e", "**casal**e"
  - No match: "casa"
- `[^`*caratteri*`]`: un carattere non nell'elenco
  - Esempio: "`casa[^a-z]`"
  - Match: "**casa!**", "**casa** dolce casa"
  - No match: "casa", "casale"

Inizio e fine stringa

- '^': inizio stringa
- '$': fine stringa
  - Esempio: "^casa$"
  - Match: "**casa**"
  - No match: "casale", "rincasa"

Stringa opzionale

- '?': zero o una volta
  - Esempio: "https?://"
  - Match: "**http://**", "**https://**"
  - No match: "httpss://"

Ripetizioni

- '*': zero o più volte
  - Esempio: "0*123"
  - Match: "**123**", "**0123**", "**000123**", "0.**123**45"
  - No match: "023", "000012"
- '+': una o più volte
  - Esempio: "0+123"
  - Match: "**0123**", "**000123**"
  - No match: "123", "023", "000012", "0.12345"

Sottoespressione

- '( )': sequenza di caratteri da considerarsi come un elemento unico
  - Esempio: "http://(www.)?example.com"
  - Match: "**http://www.example.com**", "**http://example.com**"
  - No match: "http://ww.example.com"
- Usato anche per la *substitution*

Alternativa

- '|': una sequenze tra varie possibili unico
  - Esempio: "www.example.(com|org|net)"
  - Match: "http://**www.example.com**", "**www.example.org**"
  - No match: "www.example.it"

Espressioni regolari: conclusioni

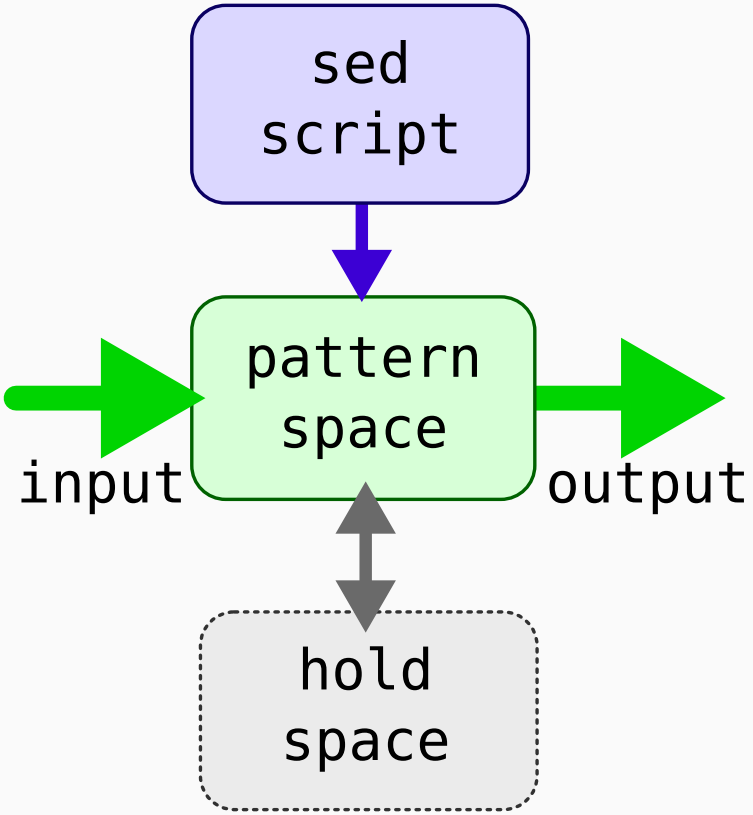
- Sono uno strumento potente, ma insidioso
- Da verificare sempre con cura
  - `grep -E '`*`regexp`*`'`
  - strumenti online (es: `http://regexr.com/`)
  - Attenzione a falsi positivi e falsi negativi

sed

sed = the Stream EDitor

Per ogni linea di input:

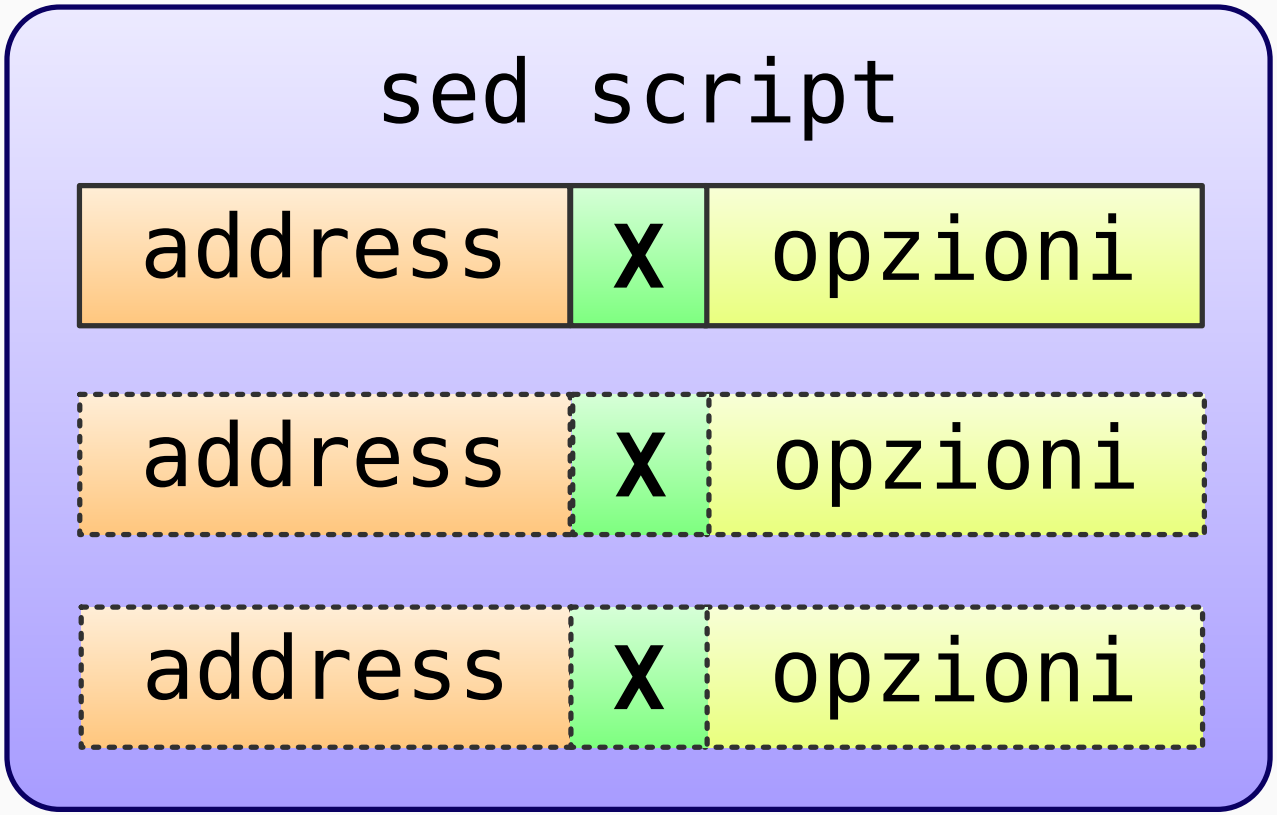
- Memorizza la riga nel *pattern space*
- Esegue il *sed script* che modifica il pattern space
- emette il *pattern space*



sed script

Sed script = una o più terne:

- address (opzionale): se corrisponde, il comando viene eseguito
- comando: una lettera
- eventuali parametri del comando



sed script

- Address
  - Numero di riga
  - Range di numeri di riga
  - `/regexp/`
- Comandi principali
  - `s/regexp/replacement/` — sostituzione
  - `d` — elimina la riga
- Demo!

awk

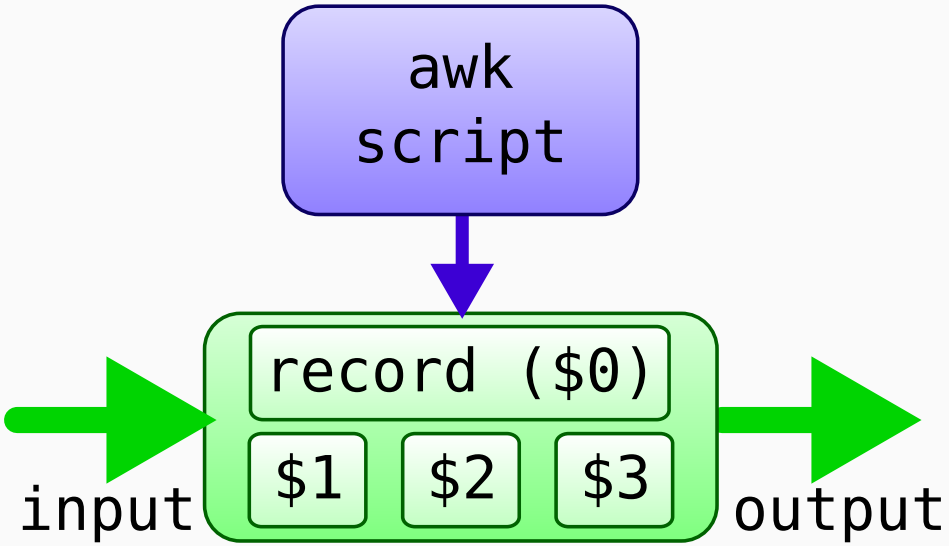
awk

- Linguaggio di programmazione progettato per estrarre ed elaborare dati
- Struttura simile a `sed`: coppie pattern + comando
- Ma il comando è un vero programma
  - linguaggio simile al C

Flusso di esecuzione

Per ogni linea di input:

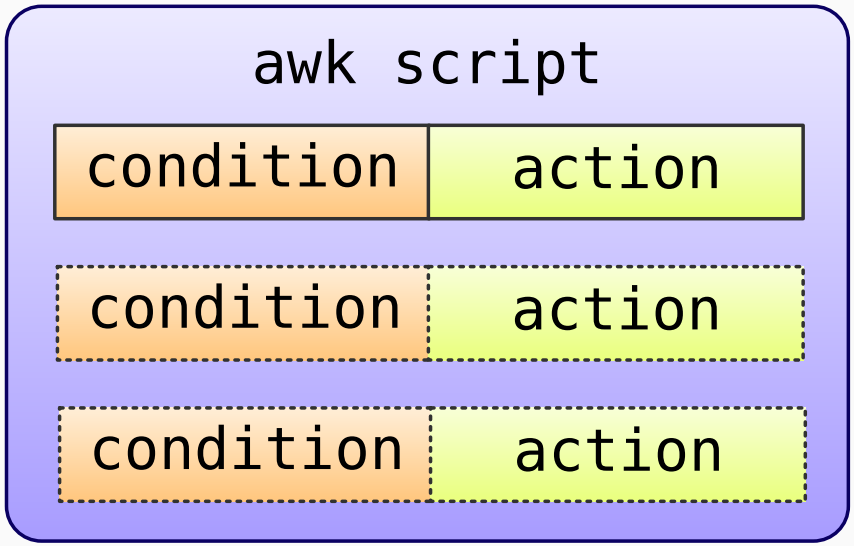
- Memorizza la riga (*record*)
  - E i singoli campi
- Esegue l'*awk script*



awk script

Sed script = una o più coppie:

```
condizione { azione }
```



Se la condizione corrisponde, l'azione viene eseguita

Azioni

- Un'azione è programma, con sintassi simile al C
- Possibilità:
  - Operatori logici
  - Espressioni matematiche
  - Variabili
  - Array associativi
  - `if`, `for`, …
  - Chiamate a funzione
  - …

Condizioni

- */regexp/*
- Qualsiasi espressione. Esempi:
  - `counter == 10` — valore di una variabile
  - `NR == 7` — settima riga
- `BEGIN`: l'azione sarà eseguita prima di leggere l'input
- `END`: l'azione sarà eseguita dopo la fine dell'input

Demo 1

Alcuni semplici programmi awk

Field e record

- awk spezza l'input in record
  - normalmente record = riga
  - Il suo contenuto viene scritto nella variabile `$0`
  - Il numero di record viene scritto in `NR`
  - Il record separator si può modificare scrivendo la variabile `RS` (default: newline)
- Spezza poi il record in field
  - normalmente: testo delimitato da spazi
  - Ogni field viene scritto nelle variabili `$1`, `$2`, `$3`, ...
  - Il numero di field viene scritto in `NF`
  - Il field separator si può modificare scrivendo la variabile `FS` (default: spazio)

Demo 2

Qualche esempio più complesso...

Conclusioni

Grazie per l'attenzione!

Domande?

luca@lucaceresoli.net

http://lucaceresoli.net