Esercizi quoting

- preparare una directory con dentro alcuni file, di nome `a`, `a1`, `a2`, `b1`, `b11` ed eseguire i test seguenti in tale directory
  - ipotizzare il risultato, verificarlo, interpretarlo
- assegnare A=a*
  - impartire `echo $A` / `echo "$A"` / `echo '$A'`
- assegnare A=[12]
  - impartire `echo $A` / `echo "$A"` / `echo '$A'`
  - impartire `echo a$A` / `echo a"$A"` / `echo a'$A'`
- assegnare A={a1,a2}
  - impartire `echo{a1,a2}` / `echo $A`
- creare una sottodir. di nome `f` e copiare lì tutti i file
  - `mkdir f ; cp * f` (c'è un errore, cosa lo causa?)
  - impartire `ls -l / ls -l *`
  - l'output è diverso? che opzione aggiunta al secondo caso lo renderebbe più simile al primo?
- come creo un file di nome `a1 a2` (singola stringa, spazio intermedio compreso)?
- come assegno tale stringa a una variabile A?
  - impartire `ls -l $A` / `ls -l "$A"`

Filtri e pipeline

qualsiasi comando unix ha a disposizione 3 file con cui comunicare con il resto del sistema:

standard input in ingresso

standard output in uscita

standard error in uscita

Filtri e pipeline

Per i comandi lanciati da un terminale, lo standard input viene agganciato alla tastiera e gli altri due al video

Filtri e pipeline

- Gli stream di un processo possono essere ridiretti verso e da file

  `comando > file` ridirige stdout verso file, troncandolo

  `comando >> file` ridirige stdout verso file, in append

  `comando 2> file` ridirige stderr verso file, troncandolo

  `comando 2>> file` ridirige stderr verso file, in append

  `comando < file` ridirige stdin da file

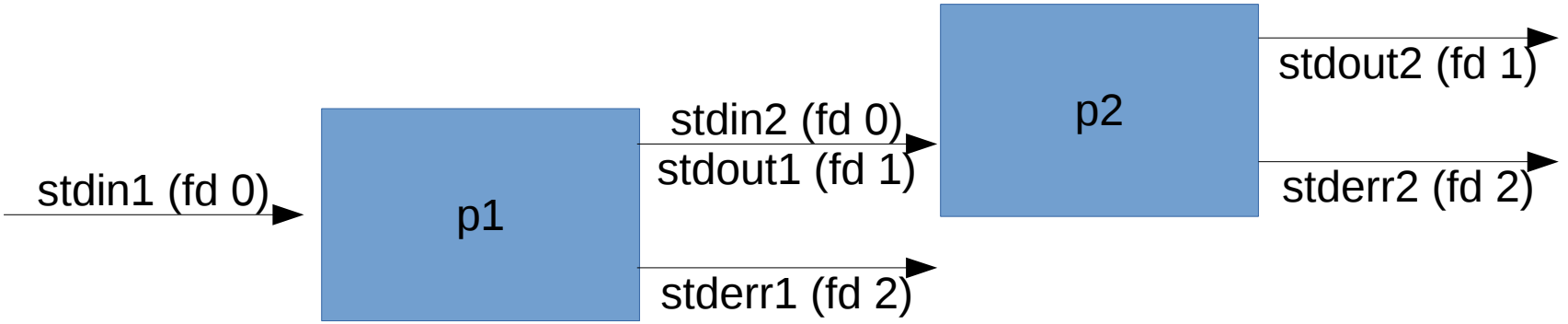
- N>&M ridirige sul file descriptor M i dati scritti da un processo sul file descriptor N, esempio:

  `comando 2>&1` ridirige stderr su stdout

Filtri e pipeline

- Un altro tipo di ridirezione è tra processi:

```
p1 | p2
```



- Quel che non è ridiretto resta dov'era: da/per il terminale
- Non limitato a due elementi

```
p1 | p2 | p3 | p4 ...
```

Filtri e pipeline

- esempi: sperimentare pipe con filtri di base (man pages!)

  cat /etc/passwd | ...
  - head
  - tail
  - sort
  - cut -c5-10
  - cut -f6 -d:
  - uniq
  - wc

Variabili e command substitution

- esempi di assegnazione di costanti a variabili
  - la shell fa espansione sulla riga?
  - ci sono tipi?
- trasformare uno stream (stdout) in una stringa

```
primidieci=$(ls -ls | sort -n | head -10)
```

test

- bash fa pochissime cose internamente
  - lancia comandi e verifica l'exit code
  - es. per testare condizioni c'è un comando esterno → **`man test`**
  - nelle versioni moderne: builtin **`[[ ]]`** → **`help [[`** o **`man bash`**
- utilizzabile per controllo di flusso

```
if comando ; then
   codice eseguito se exit true
else
   codice eseguito se exit false
fi
```

- e per cicli

```
while comando ; do
   codice eseguito fintanto che exit true
done
```

esempi

- come condiziono un'azione alla verifica che una variabile contenga un numero minore di 7?
- come eseguo un ciclo che itera su tutti i file che iniziano con a della directory corrente?
- come eseguo un ciclo che mostra lo uid dei proprietari dei file della directory corrente?
- come eseguo un ciclo infinito che ripete il corpo ogni 40 secondi?
- come eseguo un ciclo che legge da tastiera una riga e procede fintanto che l'input è diverso da "basta"?

  → **`help read`**

cicli e pipeline

- Ogni comando presente in una pipeline viene eseguito da una subshell diversa, generata da una *fork*().
- Nel caso il comando sia esterno, sostitusce il processo *bash* in seguito alla chiamata di *exec*()
- Nel caso invece sia un built-in, è naturalmente la subshell medesima (quindi un processo *bash* figlio di quello che sta interpretando la pipeline nel suo complesso) ad attuarlo
- In termini di processi cosa succede in questo esempio?

```
export COUNT=0; cat file | while read line ; do COUNT=$
(( $COUNT + 1 )) ; done ; echo $COUNT
```

*** il costrutto $(( )) viene espanso col risultato della valutazione dell'espressione contenuta tra parentesi*

cicli, pipeline e subshell

In questi casi è possibile forzare con **( )** l'esecuzione di una sequenza di comandi in una subshell: l'interprete in tal caso

non ne aprirà di superflue per i interpretare i builtin

chiuderà la subshell solo quando esplicitamente indicato

```
export COUNT=0; cat file | ( while read line ; do
COUNT=$(( $COUNT + 1 )) ; done ; echo $COUNT )
```

In questo esempio la variabile COUNT aggiornata dal ciclo *while* è la stessa usata da *echo*, poiché entrambi i comandi sono eseguiti nella subshell avviata e mantenuta grazie all'indicazione delle parentesi tonde

Esecuzione remota

- Dato un file di nome "testo" sulla macchina locale, lo si faccia remotamente ordinare alla macchina che ha meno processi in esecuzione tra quelle elencate nel file "lista", memorizzando il risultato nel file "testo.ord" in locale.
- Separare il problema in tre parti:
  - uno script "sshnum.sh" che conta i processi di una macchina passata come parametro
  - uno script "sshload.sh" che seleziona la macchina con meno processi tra quelle in lista
  - uno script "sshsort.sh" che esegue l'ordinamento del testo sulla macchina passata come parametro

  (si ricordi che i parametri passati a uno script sono disponibili ad esso come variabili numeriche $1 $2 $3 ...)
- Possibile problema: un ciclo pare fermarsi al primo elemento…
  Suggerimento: chi sta "rubando lo standard input?
- Esercizio proposto: sshload può funzionare male se una macchina di "lista" è spenta. Verificare perché e risolvere il problema.