Useless Use of *

Jan Schaumann
jschauma@netmeister.org
PGP: 136D 027F DC29 8402 7B42 47D6 7C5B 64AF AF22 6A4C

whoami

```
$ ME=$(id -un)
$ grep ${ME} /etc/passwd | cut -d: -f5
Jan Schaumann
$
```

whoami

```
$ ME=$(id -un)
$ grep ${ME} /etc/passwd | cut -d: -f5
Jan Schaumann
$ groups ${ME}
netbsd sa yahoo
$
```

whoami

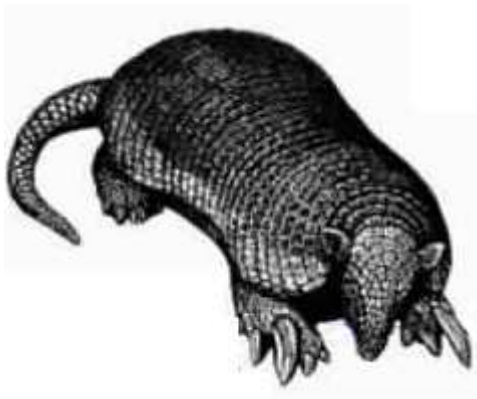
```
$ ME=$(id -un)
$ grep ${ME} /etc/passwd | cut -d: -f5
Jan Schaumann
$ groups ${ME}
netbsd sa yahoo
$
```

whoami

```
$ ME=$(id -un)
$ grep ${ME} /etc/passwd | cut -d: -f5
Jan Schaumann
$ groups ${ME}
netbsd sa yahoo
$
```

whoami

```
$ ME=$(id -un)
$ grep ${ME} /etc/passwd | cut -d: -f5
Jan Schaumann
$ groups ${ME}
netbsd sa yahoo
$
```



whoami

```
$ ME=$(id -un)
$ grep ${ME} /etc/passwd | cut -d: -f5
Jan Schaumann
$ groups ${ME}
netbsd sa yahoo
$
```

whoami

```
$ ME=$(id -un)
$ grep ${ME} /etc/passwd | cut -d: -f5
Jan Schaumann
$ groups ${ME}
netbsd sa yahoo
$
```

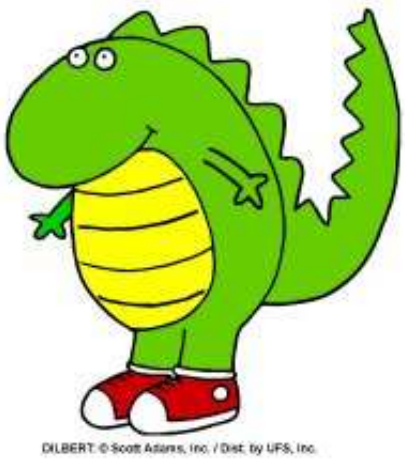
http://pipes.yahoo.com

Useless Use of... what?

Back in the day...

Useless Use of... what?

Back in the day...



Useless Use of... what?

Back in the day...

The Operator

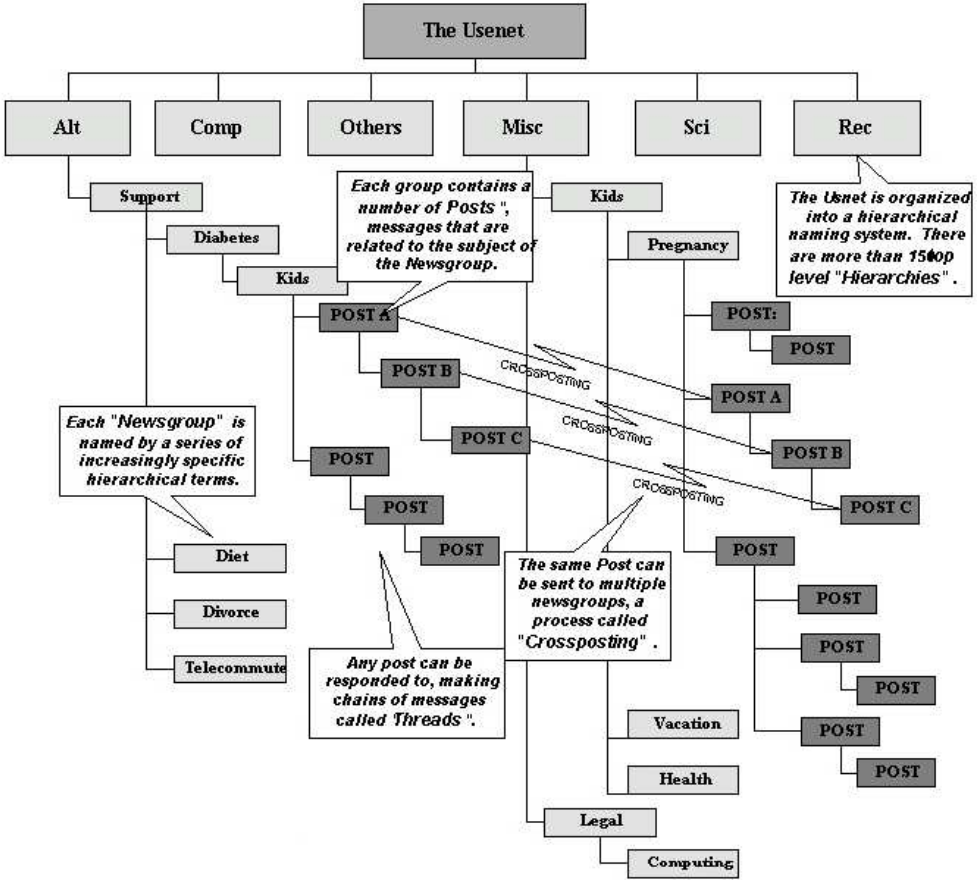
Useless Use of... what?

Back in the day...

BOFH

Useless Use of... what?

Back in the day...



Useless Use of... what?

Back in the day...

Useless Use of... what?

Back in the day...

Useless Use of... what?

Back in the day...

Useless Use of... what?

comp.unix.shell

Useless Use of... what?

I have a bunch of text files that contain strings containing /u. I no longer want them to contain /u ex: /u/appl/ should be /appl/....

Should I use awk? sed? Help!

Useless Use of... what?

```
> I have a bunch of text files that contain strings
> containing  /u.  I no longer want them to contain
> /u ex: /u/appl/  should be /appl/....
>
> Should I use awk? sed? Help!

cat file |  sed -e "s!/u/!/!"
```

Useless Use of Cat

Useful Use of Cat (?)

The obvious:

`cat` *file* `| grep` *pattern*

`cat` *file* `| awk '{ print $2; }'`

Useless Use of Cat

The obvious:

`cat` *file* `|` `grep` *pattern*

`cat` *file* `|` `awk '{ print $2; }'`

Useless Use of Cat

The obvious:

~~`cat` *file* `|` `grep` *pattern*~~

grep *pattern file*

~~`cat` *file* `|` `awk '{ print $2; }'`~~

awk '{ print $2; }' < *file*

Useful Use of Cat (?)

`cat` *file1 file2 file3* `| wc -l`
`cat` *file1 file2 file3* `| wc -w`

Useless Use of Cat

`cat` *file1 file2 file3* `| wc -l`
`cat` *file1 file2 file3* `| wc -w`

Useless Use of Cat

~~`cat` *file1 file2 file3* `|` `wc -l`~~

`awk 'END { print NR }'` *file1 file2 file3*

~~`cat` *file1 file2 file3* `|` `wc -w`~~

`awk '{w = w + NF} END { print w }'` *file1 file2 file3*

Useful Use of Cat (?)

```
cat file1 file2 file3 | grep pattern
if [ $(cat files | grep -c pattern) -gt 0 ]; then
```

Useless Use of Cat

`cat` *file1 file2 file3* `|` `grep` *pattern*

`if [ $(cat` *files* `|` `grep -c` *pattern*`) -gt 0 ]; then`

Useless Use of Cat

~~`cat` *file1 file2 file3* `|` `grep` *pattern*~~

grep -h *pattern file1 file2 file3*

`if [ $(cat` *files* `| grep -c` *pattern*`) -gt 0 ]; then`

Useless Use of Cat

~~`cat` *file1 file2 file3* `|` `grep` *pattern*~~

grep -h *pattern file1 file2 file3*

awk '/*pattern*/ **{ print }**' *file1 file2 file3*

`if [ $(cat` *files* `| grep -c` *pattern*`) -gt 0 ]; then`

Useless Use of Cat

~~`cat` *file1 file2 file3* `|` `grep` *pattern*~~

grep -h *pattern file1 file2 file3*

awk '/*pattern*/ **{ print }**' *file1 file2 file3*

~~`if [ $(cat` *files* `|` `grep -c` *pattern*`) -gt 0 ]; then`~~

if [**-n** "**$(grep -l** *pattern files*)"]; **then**

Useless Use of Cat

~~`cat` *file1 file2 file3* `|` `grep` *pattern*~~
grep -h *pattern file1 file2 file3*
awk '/*pattern*/ **{ print }**' *file1 file2 file3*

~~`if [ $(cat` *files* `|` `grep -c` *pattern*`) -gt 0 ]; then`~~
~~**if** `[ -n "$(grep -l` *pattern files*`)" ]; then`~~
if grep *pattern files* >/**dev/null 2**>&1; **then**

Useful Use of Cat

- concatenate and print files

Useful Use of Cat

- concatenate and print files (D’oh!)

```
cat * > file
```

Useful Use of Cat

- concatenate and print files (D’oh!)

```
cat * > file
```

- feed file as input to another command in a particular order

Useful Use of Cat

- concatenate and print files (D’oh!)

```
cat * > file
```

- feed file as input to another command in a particular order

```
{ echo $VAR1; cat file; cmd1; } | command
```

Useful Use of Cat

- concatenate and print files (D’oh!)

```
cat * > file
```

- feed file as input to another command in a particular order

```
{ echo $VAR1; cat file; cmd1; } | command
```

- use as a NOOP

Useful Use of Cat

- concatenate and print files (D'oh!)

```
cat * > file
```

- feed file as input to another command in a particular order

```
{ echo $VAR1; cat file; cmd1; } | command
```

- use as a NOOP

```
if condition; then
        cmd1 | cmd2 | cmd3
else
        cmd1 | cmd3
fi
```

Useful Use of Cat

- concatenate and print files (D'oh!)

```
cat * > file
```

- feed file as input to another command in a particular order

```
{ echo $VAR1; cat file; cmd1; } | command
```

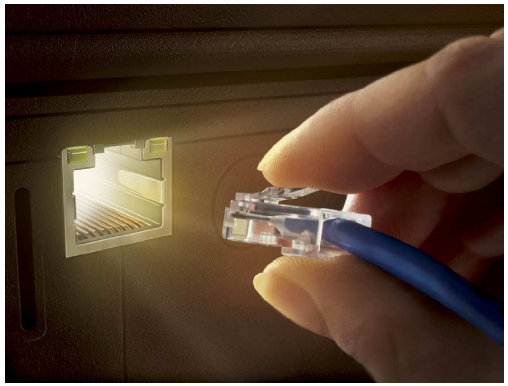
- use as a NOOP

```
if condition; then
        filter=cmd2
else
        filter=cat
fi
cmd1 | ${filter} | cmd3
```

Useful Use of Cat

CAT5

CAT6



Why bother?

Why bother?

$2.95

Why bother?

$5.90

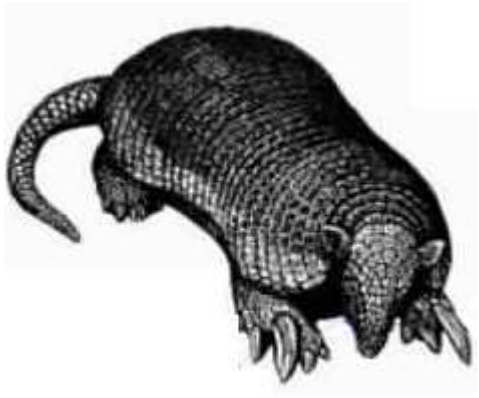
Why bother?

$11.80

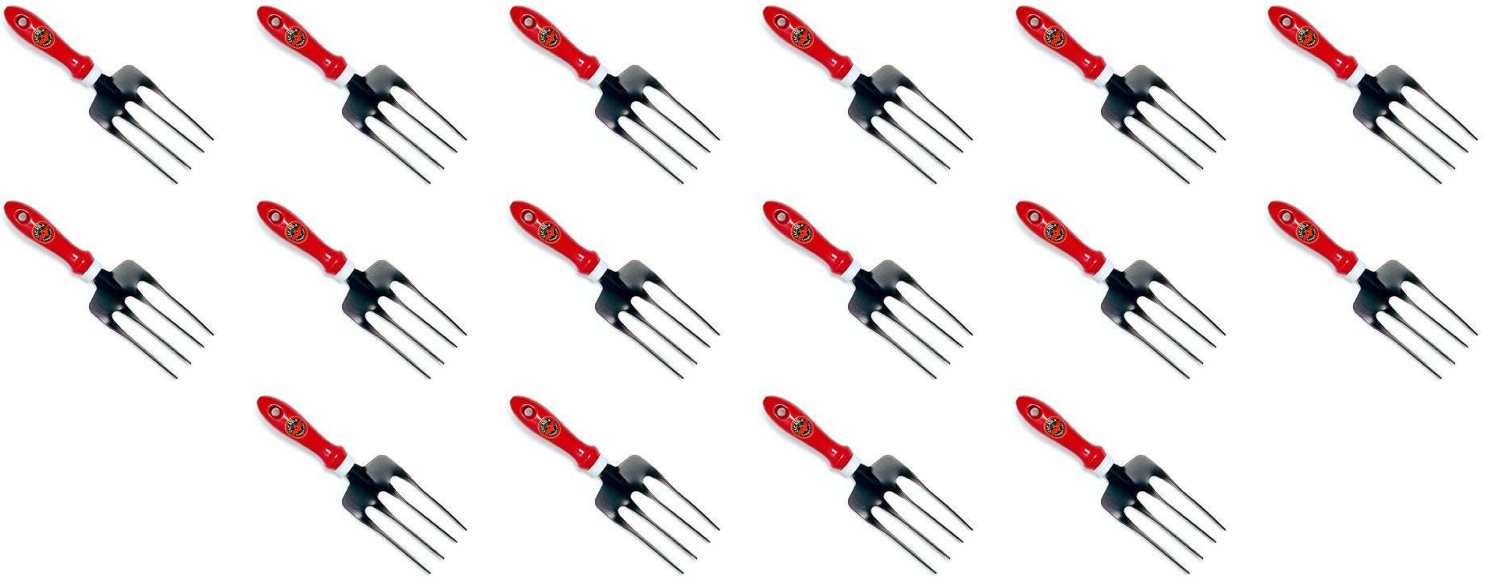
Why bother?

$23.60

Why bother?

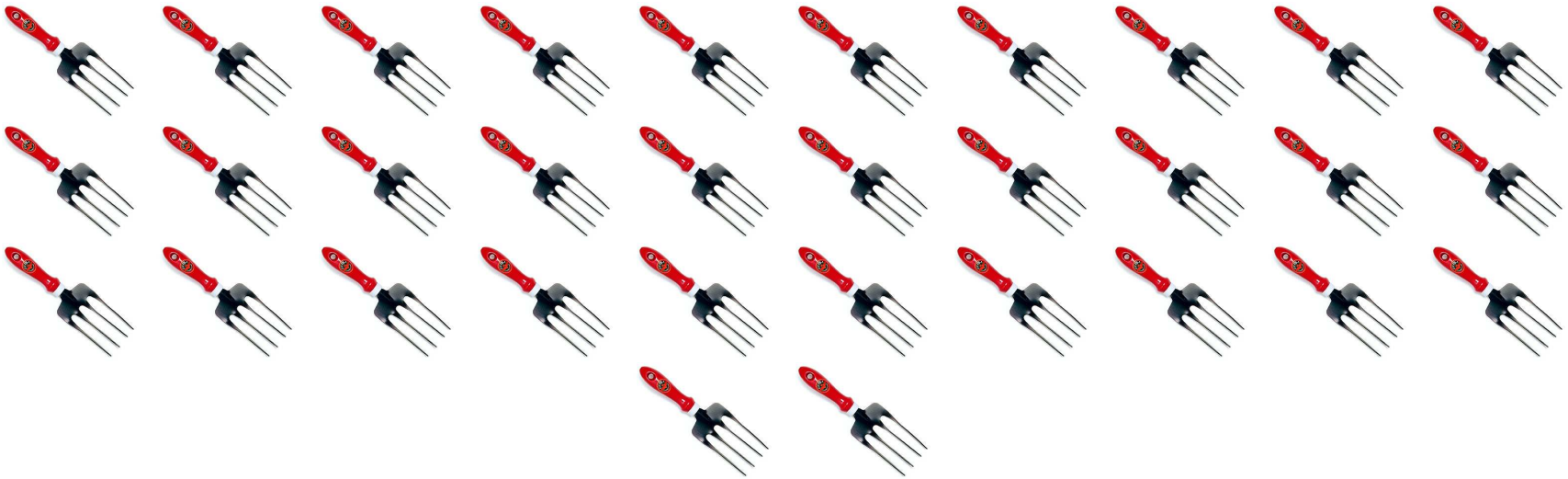


Why bother?



$47.20

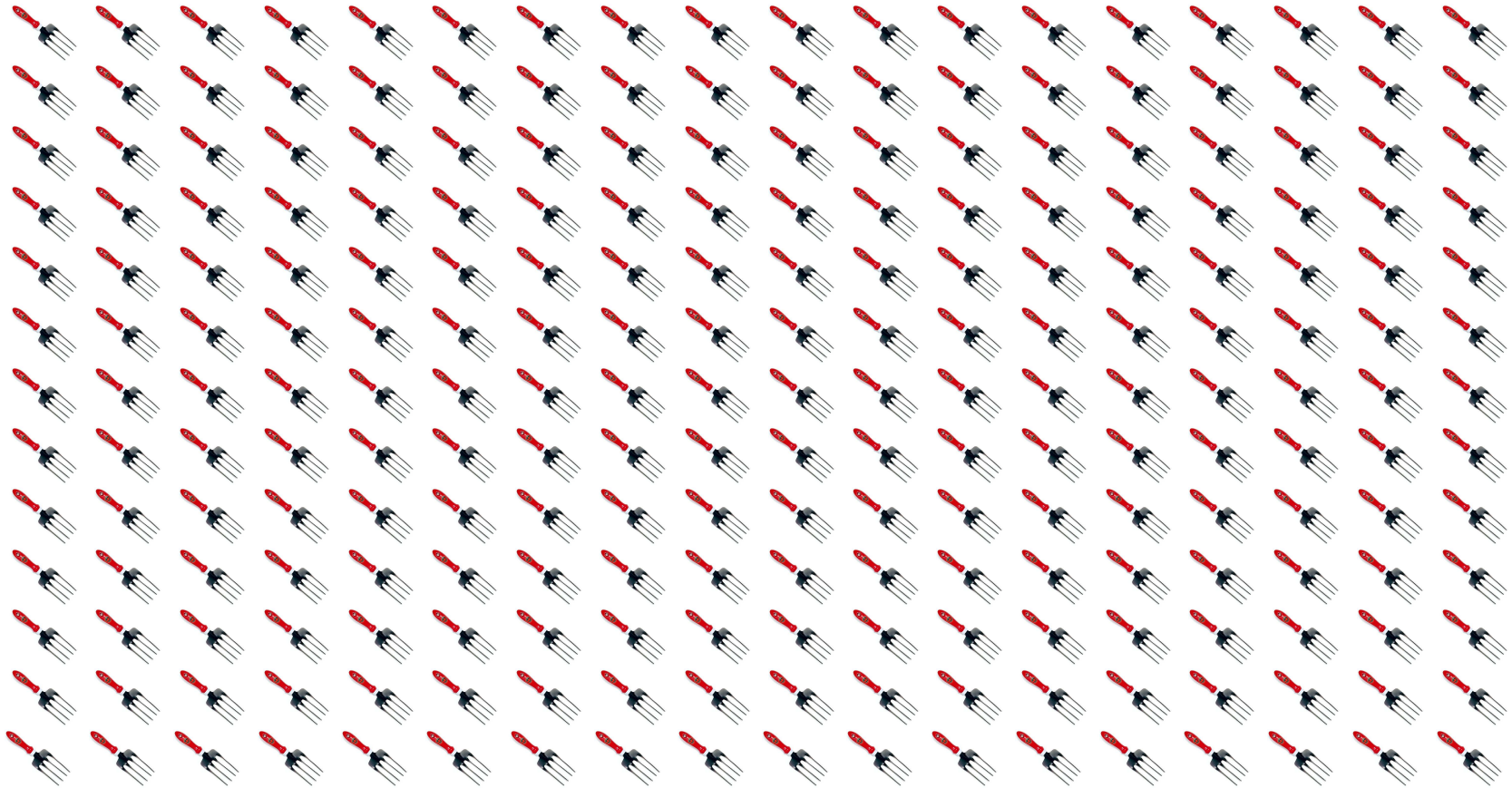
Why bother?



$94.4

Why bother?

Why bother?



But...

But...

... I run my scripts only once!

But...

... I have super fast hardware!

But...

... I have super fast hardware!

```
$ uptime
10:36AM  up 254 days, 4 users, load averages: 80.12, 75.51, 72.40
$
```

But...

... nobody else but me uses my code!

Oh, really?

Unfortunately, you really have no control over your code:

Oh, really?

Unfortunately, you really have no control over your code:

- code grows

Oh, really?

Unfortunately, you really have no control over your code:

- code grows
- code is reused

Oh, really?

Unfortunately, you really have no control over your code:

- code grows
- code is reused
- code moves with you

Oh, really?

Unfortunately, you really have no control over your code:

- code grows
- code is reused
- code moves with you
- code is left behind

Oh, really?

Unfortunately, you really have no control over your code:

- code grows
- code is reused
- code moves with you
- code is left behind

It’s alive!

Oh, really?

Oh, really?

Useless Use of *

Useless Use of Cat

Useless Use of *

Useless Use of *

Useless Use of Grep

Useless Use of Grep

Of course, **all** use of grep(1) is useless!

Useless Use of Grep

Of course, **all** use of grep(1) is useless!

`echo g/`*RE*`/p | ed -s` *file*

Useful Use of Grep (?)

```
host hostname | grep 'has address' | awk '{ print $NF }'
echo ${string} | grep 'pattern' | sed -e 's/foo/bar/'
```

Useless Use of Grep

```
host hostname | grep 'has address' | awk '{ print $NF }'
echo ${string} | grep 'pattern' | sed -e 's/foo/bar/'
```

Useless Use of Grep

~~`host hostname | grep 'has address' | awk '{ print $NF }'`~~

`host hostname | awk '/has address/ { print $NF }'`

```
echo ${string} | grep 'pattern' | sed -e 's/foo/bar/'
```

Useless Use of Grep

~~`host hostname | grep 'has address' | awk '{ print $NF }'`~~

host hostname | awk '/has address/ { print $NF }'

~~`echo ${string} | grep 'pattern' | sed -e 's/foo/bar/'`~~

echo ${string} | sed -ne '/pattern/ s/foo/bar/p'

Useful Use of Grep (?)

`grep` *pattern1 file* `...  | grep -v` *pattern2*

`grep` *pattern1 file* `| grep -v ^# | grep -v` *pattern2*

Useless Use of Grep

`grep` *pattern1 file* `...  | grep -v` *pattern2*
`grep` *pattern1 file* `| grep -v ^# | grep -v` *pattern2*

Useless Use of Grep

~~`grep` *pattern1 file ...* `|` `grep -v` *pattern2*~~

awk '/*pattern1*/ && !/*pattern2*/ **{ print }**' *file* **...**

`grep` *pattern1 file* `|` `grep -v ^#` `|` `grep -v` *pattern2*

Useless Use of Grep

~~`grep` *pattern1 file* `...` `|` `grep -v` *pattern2*~~

awk '/*pattern1*/ && !/*pattern2*/ { **print** }' *file* **...**

~~`grep` *pattern1 file* `|` `grep -v ^#` `|` `grep -v` *pattern2*~~

awk '/*pattern1*/ && !/(^#)|(*pattern2*)/ { **print** }' *file*

Useful Use of Sed (?)

```
for p in $(echo ${PATH} | sed -e 's/:/ /'); do
      ls ${p}
done
```

Useless Use of Sed

```
for p in $(echo ${PATH} | sed -e 's/:/ /'); do
      ls ${p}
done
```

Useless Use of Sed

~~for p in $(echo ${PATH} | sed -e 's/:/ /'); do~~

```
IFS=":"; for p in ${PATH}; do
      ls ${p}
done
```

Useless Use of Sed

~~for p in $(echo ${PATH} | sed -e 's/:/ /'); do~~

```
IFS=":"; for p in ${PATH}; do
      ls ${p}
done | awk 'BEGIN { srand() } { if (NR == int(rand()*100)) { print }}'
```

Sed

Useless Use of Sed

~~for p in $(echo ${PATH} | sed -e 's/:/ /'); do~~

```
IFS=":"; for p in ${PATH}; do
      ls ${p}
done | awk 'BEGIN { srand() } { if (NR == int(rand()*100)) { print }}'
```

Mknod ?

Useless Use of Sed

~~for p in $(echo ${PATH} | sed -e 's/:/ /'); do~~

```
IFS=":"; for p in ${PATH}; do
      ls ${p}
done | awk 'BEGIN { srand() } { if (NR == int(rand()*100)) { print }}'
```

Groff ?

Useless Use of Sed

~~for p in $(echo ${PATH} | sed -e 's/:/ /'); do~~

```
IFS=":"; for p in ${PATH}; do
      ls ${p}
done | awk 'BEGIN { srand() } { if (NR == int(rand()*100)) { print }}'
```

Shlock ?

Useless Use of Sed

~~for p in $(echo ${PATH} | sed -e 's/:/ /'); do~~

```
IFS=":"; for p in ${PATH}; do
      ls ${p}
done | awk 'BEGIN { srand() } { if (NR == int(rand()*100)) { print }}'
```

Route ?

Useless Use of Sed

~~for p in $(echo ${PATH} | sed -e 's/:/ /'); do~~

```
IFS=":"; for p in ${PATH}; do
      ls ${p}
done | awk 'BEGIN { srand() } { if (NR == int(rand()*100)) { print }}'
```

Dump ?

Useless Use of Sed

Useful Use of Sed (?)

```
VAR1="foo-bar-baz"
VAR2=$(echo ${VAR1} | sed -e 's/-baz//')
```

Useless Use of Sed

```
VAR1="foo-bar-baz"
VAR2=$(echo ${VAR1} | sed -e 's/-baz//')
```

Useless Use of Sed

```
VAR1="foo-bar-baz"
~~VAR2=$(echo ${VAR1} | sed -e 's/-baz//')~~
VAR2=${VAR1%-baz}
```

Useful Use of Sed (?)

Looping over values in a variable:

```
VAR1="foo-bar-baz"
for i in $(echo ${VAR1} | sed -e 's/-/ /g'); do
        the_needful $i
done
```

Useless Use of Sed

Looping over values in a variable:

```
VAR1="foo-bar-baz"
for i in $(echo ${VAR1} | sed -e 's/-/ /g'); do
IFS=-
for i in ${VAR1}; do
        the_needful $i
done
```

Useful Use of Sed (?)

Assigning variables:

```
VAR1="foo-bar-baz"
VAR_A=$(echo ${VAR} | sed -e 's/-*//')
VAR_B=$(echo ${VAR} | sed -e 's/[^-]*-\([^-]*\)-.*/\1/')
VAR_C=$(echo ${VAR} | sed -e 's/.*-//')
```

Useless Use of Sed

Assigning variables:

```
VAR1="foo-bar-baz"
VAR_A=$(echo ${VAR} | sed -e 's/-*//')
VAR_B=$(echo ${VAR} | sed -e 's/[^-]*-[^-]-.*/\1/')
VAR_C=$(echo ${VAR} | sed -e 's/.*-//')
```

```
IFS=-
set -- ${VAR1}
VAR_A="${1}"
VAR_B="${2}"
VAR_C="${3}"
```

Useless Use of ...

Dude, `IFS` + `shell` is Teh Roxor!!!1
Look, Ma: Reading a CSV with Shell Only!

```
IFS=","
while read -r field1 waste field3 field4 waste; do
        echo "${field1}: ${field4} --> ${field3}"
done <file
```

Useless Use of Shell Only

```
$ cat >script.sh
IFS=","
while read -r waste field2 field3 waste; do
        echo "${field1}: ${field4} --> ${field3}"
done <file
^D
$ wc -l file
  111061 file
$ time sh script.sh >/dev/null
   12.33s real     3.30s user     8.54s system
$
```

Useless Use of Shell Only

Awk to the rescue!

```
$ cat >script2.sh
awk -F"," '{ print $1 ": " $4 " --> " $3 }' <file
^D
$ time sh script2.sh >/dev/null
    1.08s real     1.08s user     0.00s system
$
```

Useful Use of Ls (?)

```
for file in $(ls *pattern*); do
        the_needful
done
```

Useless Use of Ls

~~`for file in $(ls *`*pattern*`*`); do`~~

```
for file in *pattern*; do
        the_needful
done
```

Useful Use of Wc (?)

```
VAR=$(cat file | wc -l | sed -e 's/ *//g)'
```

Useful Use of Wc (?)

~~`VAR=$(cat` *file* `| wc -l | sed -e 's/ *//g')`~~

`VAR=$(wc -l <`*file*`| sed -e 's/ *//g)'`

Useless Use of ... ?

~~`VAR=$(cat` *file* `| wc -l | sed -e 's/ *//g')`~~

~~`VAR=$(wc -l <`*file*`| sed -e 's/ *//g)'`~~

Useless Use of ... ?

~~`VAR=$(cat` *file* `| wc -l | sed -e 's/ *//g')`~~

~~`VAR=$(wc -l <`*file*`| sed -e 's/ *//g)'`~~

Useless Use of ... ?

~~`VAR=$(cat` *file* ~~`| wc -l | sed -e 's/ *//g')`~~

~~`VAR=$(wc -l <`*file*~~`| sed -e 's/ *//g)'`~~

`VAR=$(awk 'END { print NR }'` *file*`)`

Useless Use of Sed

~~VAR=$(cat *file* | wc -l | sed -e 's/ *//g')~~

~~VAR=$(wc -l <*file*| sed -e 's/ *//g)'~~

VAR=$(wc -l <*file*)

Useless Use of Sed

~~VAR=$(cat *file* | wc -l | sed -e 's/ *//g')~~

~~VAR=$(wc -l <*file*| sed -e 's/ *//g)'~~

```
VAR=$(wc -l <file)
echo "${VAR}"
```

Useless Use of Sed

~~VAR=$(cat *file* | wc -l | sed -e 's/ *//g')~~

~~VAR=$(wc -l <*file*| sed -e 's/ *//g)'~~

VAR=$(wc -l <*file*)

~~echo "${VAR}"~~

echo ${VAR}

Useful Use of Head (?)

```
command1 | head -1 | sed -e 's/pattern/string/'

command1 | head -10 | sed -e 's/pattern/string/'
```

Useless Use of Head

~~`command1 | head -1 | sed -e 's/`*pattern*`/`*string*`/'`~~

`command1 | sed -e 's/`*pattern*`/`*string*`/;q'`

`command1 | head -10 | sed -e 's/`*pattern*`/`*string*`/'`

Useless Use of Head

~~command1 | head -1 | sed -e 's/*pattern*/*string*/'~~
command1 | sed -e 's/*pattern*/*string*/;q'

~~command1 | head -10 | sed -e 's/*pattern*/*string*/'~~
command1 | awk '{ if (NR <= 10) { print gensub("*pattern*","*string*",0); } }'

Useful Use of Tail (?)

```
command1 | tail -1 | sed -e 's/pattern/string/'
```

Useless Use of Tail

~~`command1 | tail -1 | sed -e 's/`*pattern*`/`*string*`/'`~~

`command1 | awk 'END { print gensub("`*pattern*`","`*string*`",0); exit; }'`

Useful Use of Tail (?)

```
command1 | tail -10 | sed -e 's/pattern/string/'
```

Useful Use of Tail

`command1 | tail -10 | sed -e 's/`*pattern*`/`*string*`/'`

Useful Use of Expr (?)

```
echo $(expr $i + $i)
```

Useless Use of Expr

~~`echo $(expr $i + $i)`~~

```
echo $(( $i + $i ))
```

Useless Use of Expr

~~`echo $(expr $i + $i)`~~

```
echo $(( $i + $i ))
echo $(( $i << 1 ))
```

This even lets you do binary manipulation (binary and, or, xor, not, leftshift, rightshift).

Useless Use of Expr

~~`echo $(expr $i + $i)`~~

```
echo $(( $i + $i ))
echo $(( $i << 1 ))
```

This even lets you do binary manipulation (binary and, or, xor, not, leftshift, rightshift).

```
$ x=5
$ y=12
$ x=$(( ${x} ^ ${y} ))
$ y=$(( ${x} ^ ${y} ))
$ x=$(( ${x} ^ ${y} ))
$ echo "x=${x}; y=${y}"
x=12; y=5
$
```

Most Egregiously Useless Use of Perl

```
perl -e "print \"y\\nn\\n\";" | cmd
```

Most Egregiously Useless Use of Perl

~~perl -e "print \"y\\nn\\n\";" | cmd~~

```
( echo y; echo n; ) | cmd
```

Most Egregiously Useless Use of Perl

~~perl -e "print \"y\\nn\\n\";" | cmd~~

~~(echo y; echo n;) | cmd~~

```
{ echo y; echo n; } | cmd
printf "y\nn\n" | cmd
```

Shell Coding: Performance

- avoid file I/O
- avoid Useless Use of *:
  - use builtins when you can
  - avoid builtins when it makes sense
  - use the right tool for the job
- avoid spawning subshells (+1 fork) or pipes (n+1 forks)

Shell Coding: Performance

- avoid file I/O
- avoid Useless Use of *:
  - use builtins when you can
  - avoid builtins when it makes sense
  - use the right tool for the job
- avoid spawning subshells (+1 fork) or pipes (n+1 forks)

`cut -f2` vs. `awk '{print $2}'`

```
$ stat -f "%z     %N" /usr/bin/awk /usr/bin/cut
133487  /usr/bin/awk
12590   /usr/bin/cut
$
```

Testing Shell Code

- make your code modular
- clearly define each function:
  - pre-conditions
  - valid input
  - post-conditions
- prepare valid input and desired output of each function
- prepare invalid input and desired output of each function

Shell Coding Style

Writing Shell Code is just like writing any other code:

- `set -eu` (perl-mongers: think `-w` and `use strict;`)

Shell Coding Style

Writing Shell Code is just like writing any other code:

- `set -eu` (perl-mongers: think `-w` and `use strict;`)

```
$ cat >script.sh
set -u
echo ${FOO}
echo "done"
^D
$ sh script.sh
script.sh: FOO: parameter not set
$ echo $?
2
$
```

Shell Coding Style

Writing Shell Code is just like writing any other code:

- `set -eu` (perl-mongers: think `-w` and `use strict;`)

```
$ cat >script.sh
set -e
ls /nowhere
echo "done"
^D
$ sh script.sh
ls: /nowhere: No such file or directory
$
```

Shell Coding Style

Writing Shell Code is just like writing any other code:

- `set -eu` (perl-mongers: think `-w` and `use strict;`)

```
$ cat >script.sh
set -e
if ! ls /nowhere; then
        echo "ls failed"
fi
echo "done"
^D
$ sh script.sh
ls: /nowhere: No such file or directory
ls failed
done
$
```

Shell Coding Style

Writing Shell Code is just like writing any other code:

- `set -eu` (perl-mongers: think `-w` and `use strict;`)
- use `readonly` variables

```
$ cat >script.sh
readonly VAR="foo"
VAR="bar"
echo ${VAR}
^D
$ sh script.sh
script.sh: VAR: is read only
$
```

Shell Coding Style

Writing Shell Code is just like writing any other code:

- `set -eu` (perl-mongers: think `-w` and `use strict;`)
- use `readonly` variables
- use `local` variables

Shell Coding Style: local variables

```
$ cat s.sh
func () {
        input="${1}"
        # do something with ${input}
}
$ cat script.sh
. ./s.sh
input="var1 var2 var3"
for var in ${input}; do
        func "${var}"
done
echo ${input}
$ sh script.sh
var3
$
```

Shell Coding Style: local variables

```
$ cat s.sh
func () {
        local input="${1}"
        # do something with ${input}
}
$ cat script.sh
. ./s.sh
input="var1 var2 var3"
for var in ${input}; do
        func "${var}"
done
echo ${input}
$ sh script.sh
var1 var2 var3
$
```

Shell Coding Style

Writing Shell Code is just like writing any other code:

- `set -eu` (perl-mongers: think `-w` and `use strict;`)
- use `readonly` variables
- use `local` variables
- comment your code

Shell Coding Style

Writing Shell Code is just like writing any other code:

- `set -eu` (perl-mongers: think `-w` and `use strict;`)
- use `readonly` variables
- use `local` variables
- comment your code

Bad:

```
# this adds 1 to num
num=$(( ${num} + 1 ))
```

Shell Coding Style

Writing Shell Code is just like writing any other code:

- `set -eu` (perl-mongers: think `-w` and `use strict;`)
- use `readonly` variables
- use `local` variables
- comment your code

Better:

```
input="pair1l:pair1r pair2l:pair2r"
# extract first pair in input string, throw away rest
p1l=${input%%:*}
rest=${input#*:}
p1r=${rest% *}
```

Shell Coding Style

Writing Shell Code is just like writing any other code:

- `set -eu` (perl-mongers: think `-w` and `use strict;`)
- use `readonly` variables
- use `local` variables
- comment your code
- write your code in a modular way

Shell Coding Style

Writing Shell Code is just like writing any other code:

- `set -eu` (perl-mongers: think `-w` and `use strict;`)
- use `readonly` variables
- use `local` variables
- comment your code
- write your code in a modular way
- write test cases for your shell code

Shell Coding Style

Writing Shell Code is just like writing any other code:

- `set -eu` (perl-mongers: think `-w` and `use strict;`)
- use `readonly` variables
- use `local` variables
- comment your code
- write your code in a modular way
- write test cases for your shell code
- be consistent in your style

Shell Coding Style

Writing Shell Code is just like writing any other code:

- `set -eu` (perl-mongers: think `-w` and `use strict;`)
- use `readonly` variables
- use `local` variables
- comment your code
- write your code in a modular way
- write test cases for your shell code
- be consistent in your style
- be consistent in your user interface

Shell Coding Style

Writing Shell Code is just like writing any other code:

- `set -eu` (perl-mongers: think `-w` and `use strict;`)
- use `readonly` variables
- use `local` variables
- comment your code
- write your code in a modular way
- write test cases for your shell code
- be consistent in your style
- be consistent in your user interface
- be willing to sacrifice performance for readability

Shell Coding: Performance vs. Readability

```
awk -F: -v ME="${ME}" '{ if ($0 ~ ME) { print $5 }}' file
```

vs.

```
grep ${ME} file | cut -d: -f5
```

```
awk '{w = w + NF} END { print w }' file1 file2 file3
```

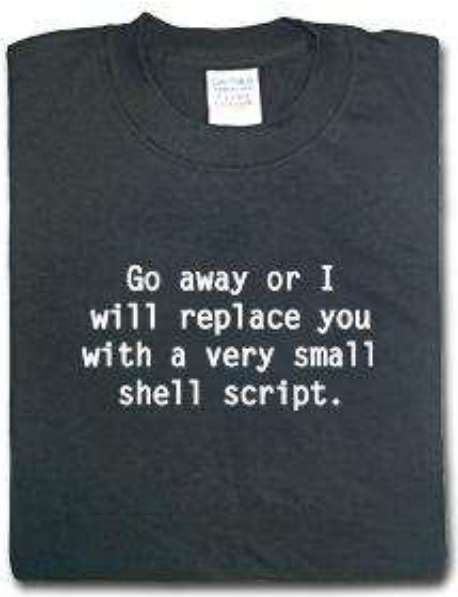
vs.

```
cat file1 file2 file2 | wc -w
```

The KISS Principle

K*eep* **I***t* **S***imple,* **S***tupid!*

Useless Use of time (?)



Thanks!