bash

by example

Matt Might

University of Utah

matt.might.net

bash

by counterexample

Matt Might

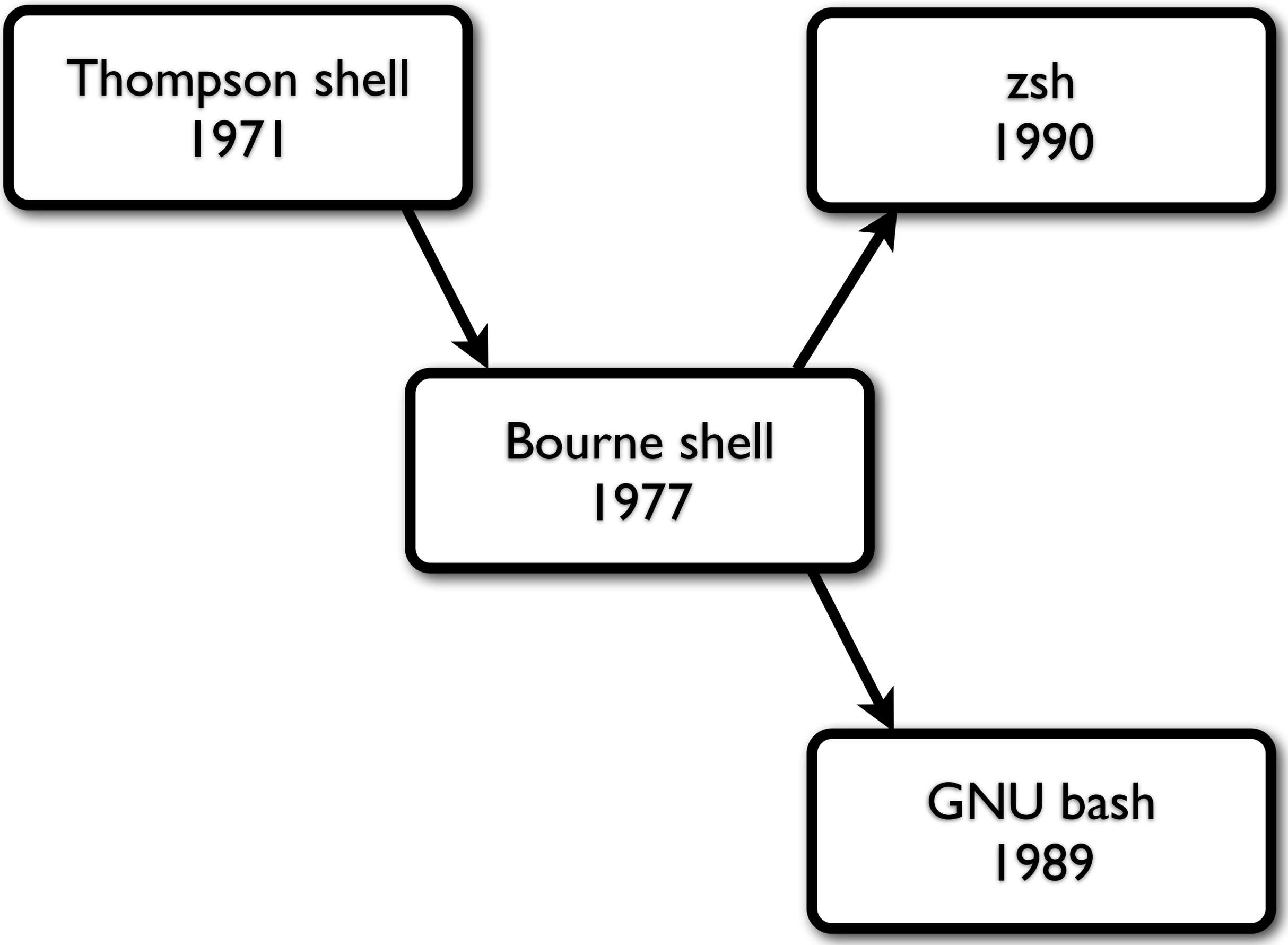
University of Utah

matt.might.net

blog pointers

- Post on starting in Unix
- Post on customizing Unix
- Post on relational scripts

Origins


Thompson shell
1971
zsh
1990
Bourne shell
1977
GNU bash
1989

Design anti-pattern:

Evolution from interaction

Design anti-pattern:

“backslash hell”

Nobody really knows what the Bourne shell's grammar is.
Tom Duff

Design anti-pattern:

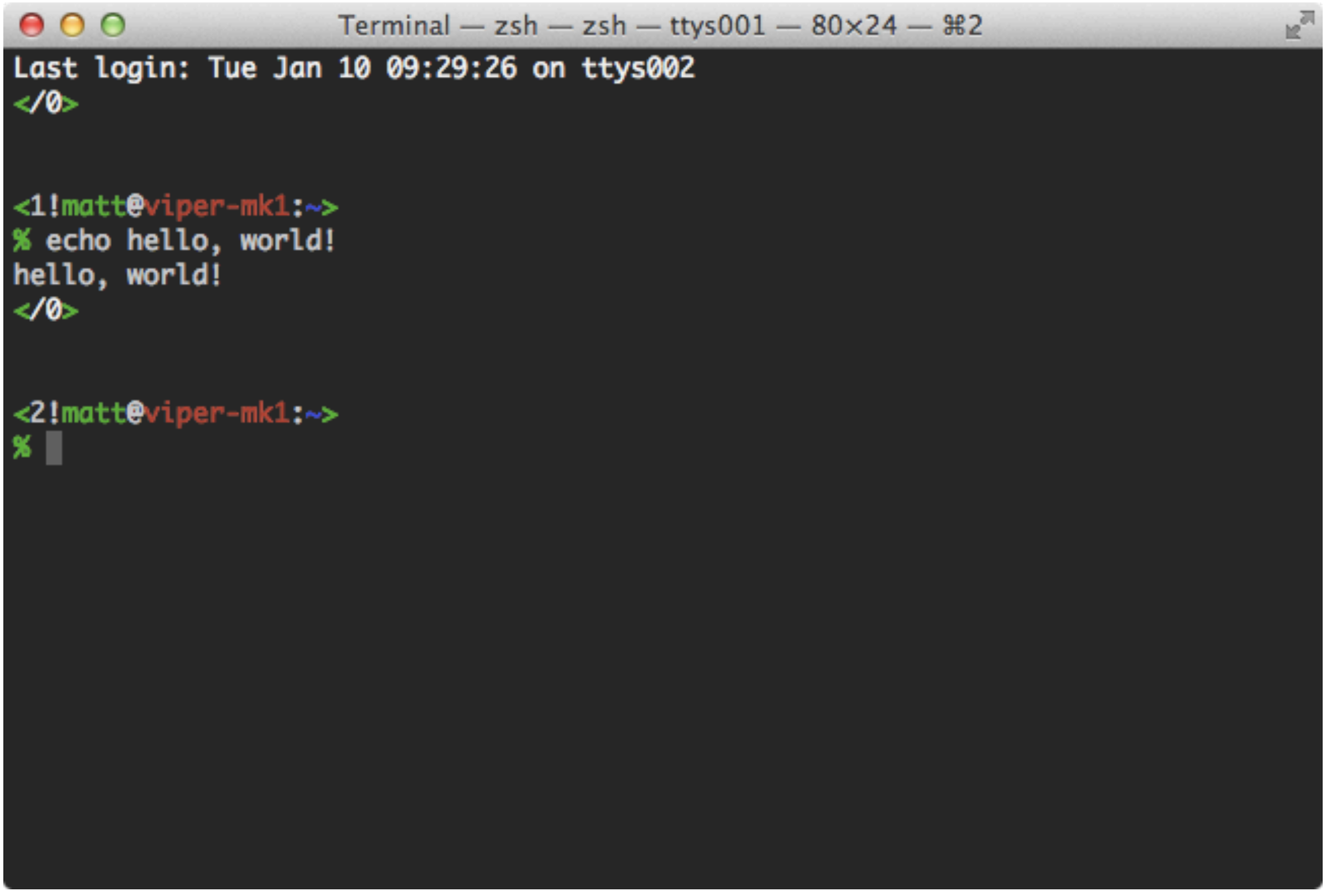
Irregular syntax

Design anti-pattern:

Process scope

But, heck, it's useful.

It’s not going anywhere.


Terminal — zsh — zsh — ttys001 — 80×24 — ⌘2
Last login: Tue Jan 10 09:29:26 on ttys002
</0>
<1!matt@viper-mk1:~>
% echo hello, world!
hello, world!
</0>
<2!matt@viper-mk1:~>
%

Variables/Arrays

foo=3

```
foo = 3
```

```
(( foo = 3 ))
```

```
echo $foo
```

```
foo=$bar; echo $foo
```

```
foo=“bar is $bar”; echo $foo
```

```
echo ${foo}
```

```
echo ${foo[0]}
```

```
foo[0]=“SLDI is great.”; echo $foo
```

```
foo[“whiz”]=“SLDI is great.”; echo $foo
```

```
foo[1]=“SLDI is great.”; echo $foo
```

```
foo=(el1 el2 el3); echo $foo
```

```
foo=(el1 el2 el3); echo ${foo[@]}
```

```
foo=(1 2)
(( bar = foo ))
bar[0]=10
echo ${foo[0]}
```

```
foo=(1 2)
(( bar = foo ))
bar[0]=10
echo ${bar[@]}
```

```
echo ${#foo}
```

```
foo=“I’m matt”; echo ${foo/matt/bill}
```

```
foo=“I’m matt”; echo ${foo/matt}
```

```
foo=‘‘I’m mattmatt’’; echo ${foo/matt/bill}
```

```
foo=‘‘I’m mattmatt’’; echo ${foo//matt/bill}
```

```
for f in music/*.mp3;
do
  echo mv $f ${f// /_}
done
```

```
echo ${foo-bar}
```

```
echo ${username-`whoami`}
```

```
echo ${foo=“x”}
```

```
foo=30
echo ${foo+10}
```

```
echo ${foo?boom}
```

```
: ${1?“usage: argument expected”}
```

```
: ${CLASSPATH?"error: no CLASSPATH"}
```

```
minipath="/usr/bin:/bin:/sbin"
echo ${minipath#/usr}
echo ${minipath#*/bin}
echo ${minipath##*/bin}
```

```
minipath="/usr/bin:/bin:/sbin"
echo ${minipath%/usr*}
echo ${minipath%%/bin*}
```

```
for f in *.sh;
do
 mv $f ${f%.sh}.bash
done
```

```
foobar=3
foobaz=4

echo ${!foo*}
```

```
echo ${foo:3:10}
```

```
echo $0
```

```
echo $1
```

```
echo $10
```

```
echo ${10}
```

```
echo $?
```

```
echo $!
```

```
echo $_
```

```
export foo
```

Pipes and redirection

```
cat /etc/passwd > passwd2
```

```
grep DocumentRoot < /etc/apache2/httpd.conf
```

```
echo hacker:0:0 >> /etc/passwd
```

```
cat /etc/passwd | awk -F “:” ‘{ print $1 }’
```

```
cmd1 < f1 | cmd2 | cmd3 > f2
```

```
find . | grep foo 1> out
```

```
find . | grep foo 2> out
```

```
find . | grep foo 2>&1
```

```
find . | grep foo &> out
```

```
exec 6>&1

exec > LOG

exec 1>&6
```

```
cat <<BlahBlah
I’m putting text here!

BlahBlah
```

```
grep -n . <(cat /etc/passwd)
```

```
echo <(cat /etc/passwd)
```

Design principle:

Make I/O easy.

Processes

```
sort /usr/share/dict &
```

wait

```
wait $!
```

Strings and quoting

```
echo ‘‘Hello, $world’’
```

```
echo ‘Hello, $world’
```

```
echo `Hello, $world`
```

```
accounts=`cat /etc/passwd`
```

```
accounts=$(cat /etc/passwd)
```

```
accounts=`</etc/passwd`
```

$() *versus* `` ?

Design principle:

Quotes should nest.

“…” versus "…"

Globs

```
echo foo.*
```

```
echo foo*bar
```

```
echo foo?bar
```

```
echo [fh]oo
```

```
echo [a-z]*
```

```
echo {a,b,c}
```

```
echo {a,b,c}{x,y,z}
```

Expressions

```
expr 3 + 20
```

```
expr 3 * 20
```

```
expr 3 \* 20
```

let ‘‘x=3+20*a’’

```
(( x = 3 + 20 * a ))
```

Control

```
if true; then
  echo "It's not false."
fi
```

```
if true
then
  echo "It's not false."
fi
```

```
while read a
do
  echo $a
done < input
```

```
for a in *.mp3;
do
 echo $a : `md5sum $a`
done
```

```
function name {
  ...
}
```

```
name () {
  ...
}
```

Tests

```
[ 10 -lt $foo ]
```

```
(( 10 < foo ))
```

```
[ -e $f ]
```

command1 && command2

command1 || *command2*

! *command*

Pitfalls

```
cp $file $target
```

```
cp ‘‘$file’’ ‘‘$target’’
```

```
cp -- ‘‘$file’’ ‘‘$target’’
```

```
rm -rf $(dirname ‘‘$f’’)
```

```
rm -rf ‘‘$(dirname ‘‘$f’’)’’
```

```
if [ $1 = foo ]; then
  echo first arg is foo
fi
```

```
if [ ‘‘$1’’ = foo ]; then
  echo first arg is foo
fi
```

```
for f in $(ls evildir/*);
do
  echo $f
done
```

```
for f in evildir/*;
do
  echo $f
done
```

```
for f in $(find evildir);
do
   echo $f
done
```

```
find evildir -exec echo "properly: " {} \;
```

```
count=0

for l in $(< /etc/hosts)
do
  echo "$count $l"
  (( count = count + 1 ))
done | sort -rn

echo $count # prints 0
```

```
count=0

for l in $(< /etc/hosts)
do
  echo "$count $l"
  (( count = count + 1 ))
done > tmp; sort -rn tmp

echo $count
```

```
for i in *.dat
do
  echo $i
done
```

```
for i in *.dat
do
   [[ -f $i ]] || continue
   echo $i
done
```

Files named `rm`, `-f` and `*`

```
echo $foo
```

```
printf ‘‘%s\n’’ ‘‘$foo’’
```

```
cd tmp; rm -f *
```

```
cd tmp || exit 1; rm -f *
```

```
for arg in $*;
```

```
for arg in $@;
```

```
for arg in ‘‘$@’’;
```

Questions?