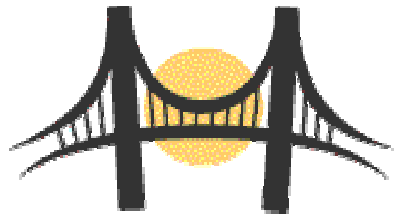
p4 add
p4 admin
p4 branch
p4 branches
p4 change
p4 changes
p4 client
p4 clients
p4 counter
p4 counters
p4 delete
p4 depot
p4 depots
p4 describe
p4 diff
p4 diff2
p4 dirs
p4 edit
p4 filelog
p4 files
p4 fix
p4 fixes
p4 flush
p4 fstat
p4 group
p4 groups
p4 have
p4 help
p4 info
p4 integrate
p4 integrated
p4 job
p4 jobs
p4 jobspec
p4 label
p4 labels
p4 labelsync
p4 lock
p4 logger
p4 obliterate
p4 opened
p4 passwd
p4 print
p4 protect
p4 rename
p4 reopen
p4 resolve
p4 resolved
p4 revert
p4 review
p4 reviews
p4 set
p4 submit
p4 sync
p4 triggers
p4 typemap
p4 unlock
p4 user
p4 users
p4 verify
p4 where

Getting Started with Jam/MR



A Tutorial

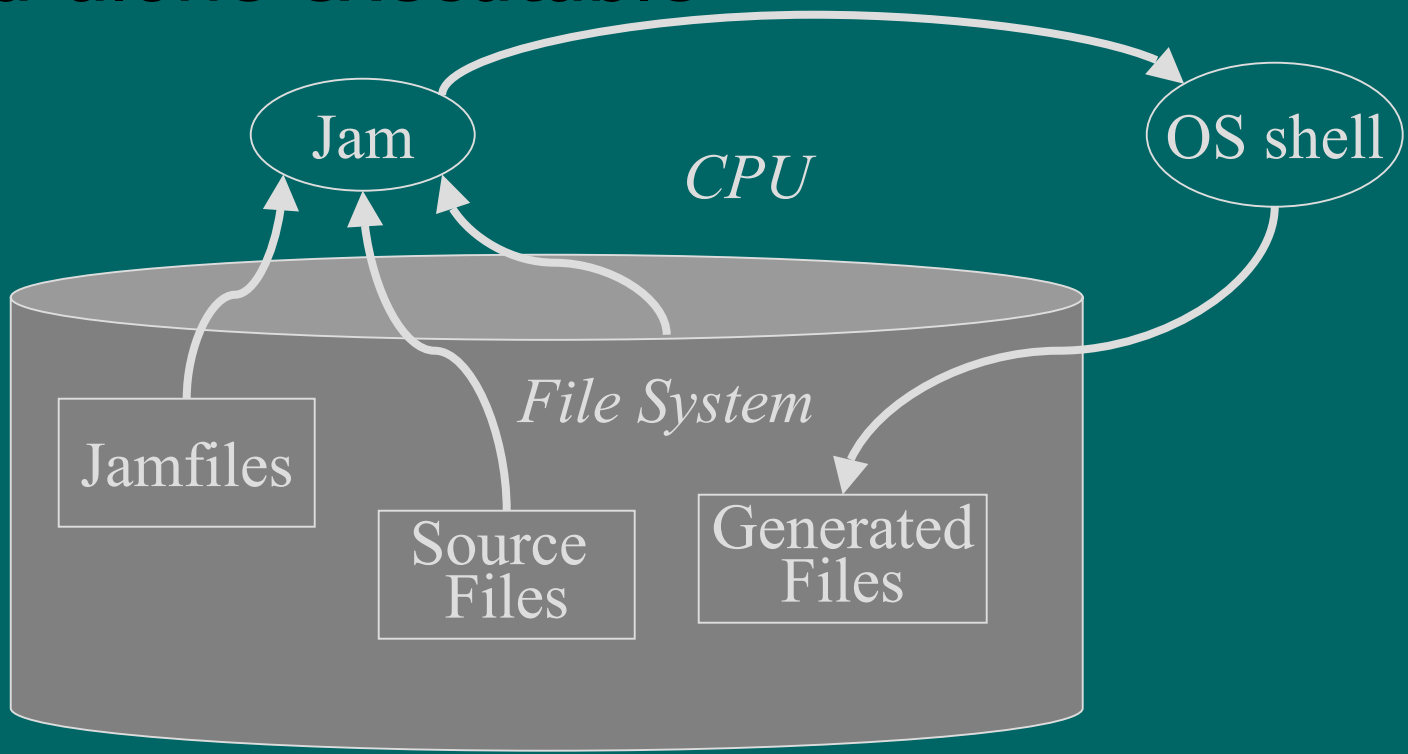
PERFORCE

Overview

- How Jam Works
- The Jam Language
- A Working Example

How Jam Works

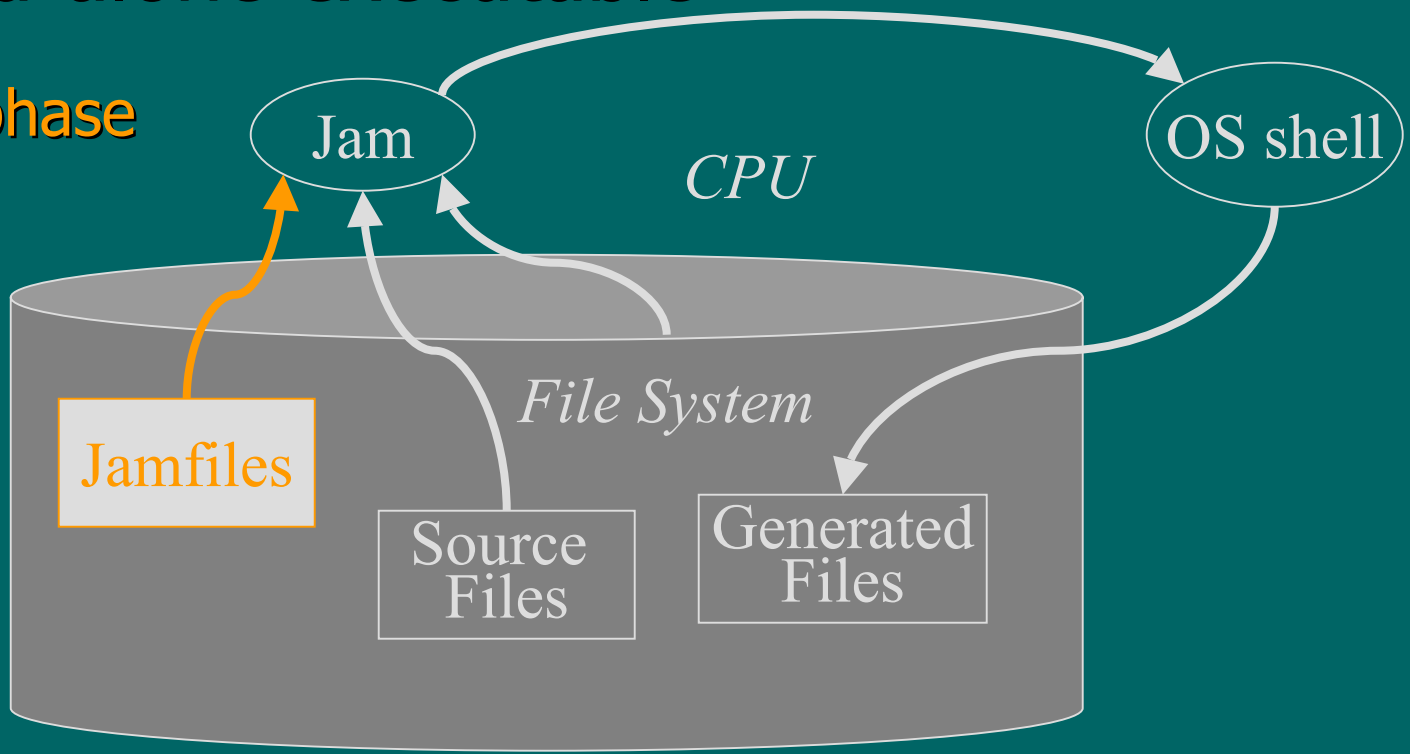
- Stand-alone executable



How Jam Works

- Stand-alone executable

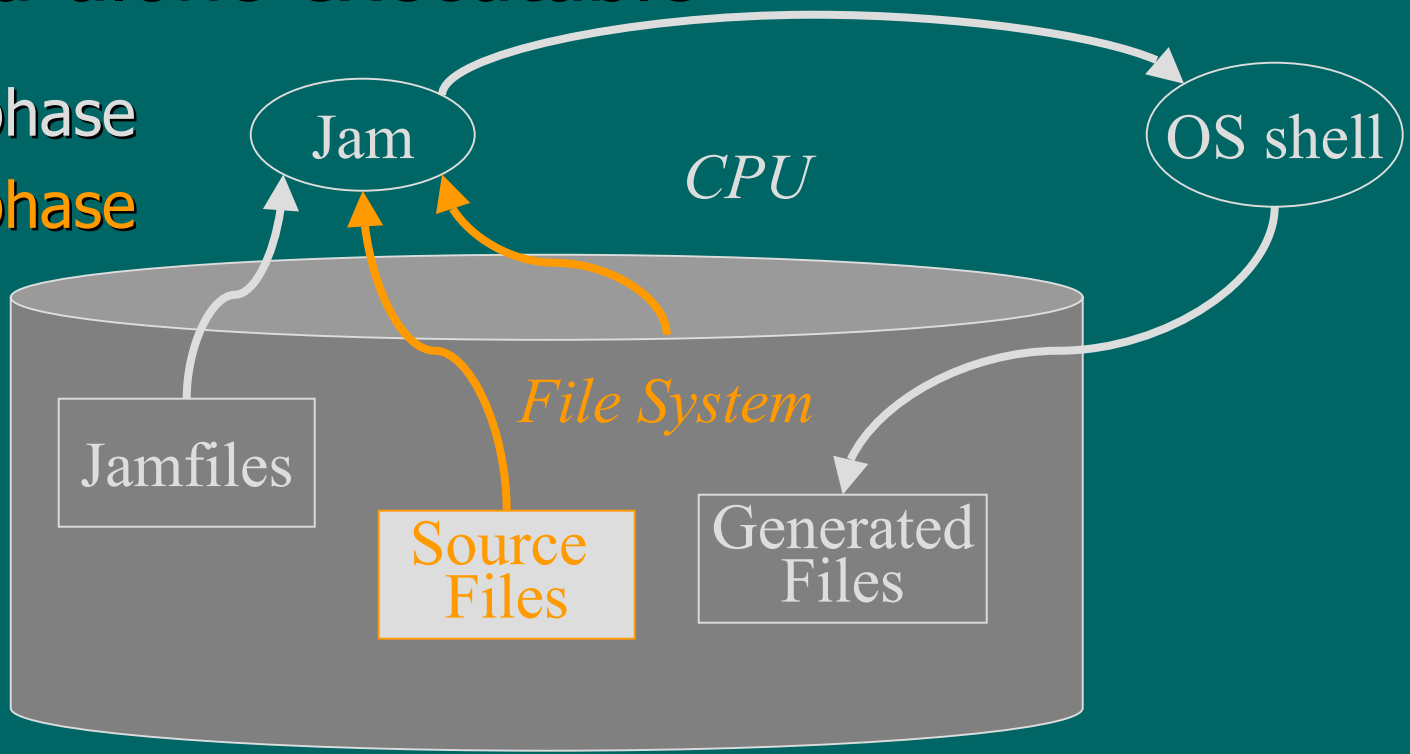
parsing phase



How Jam Works

- Stand-alone executable

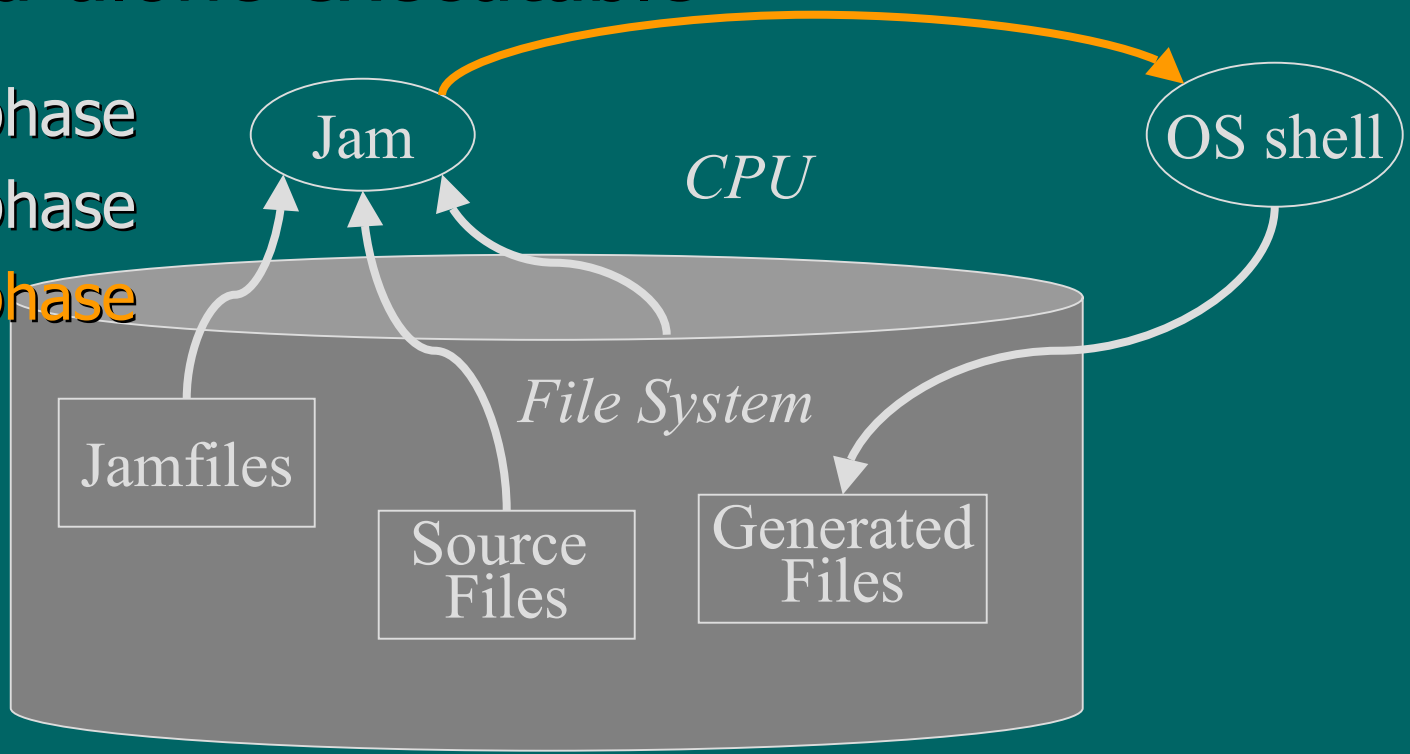
parsing phase
binding phase



How Jam Works

- Stand-alone executable

parsing phase
binding phase
updating phase



Running Jam

```
jam  [options…]  [targets…]
```

- Useful options:
  - `-n`
  - `-d2, -d5`
  - `-a`
  - `-f`*`yourfile`*

The Jam Language

- Syntax
- Variables
- Rules
- Actions
- Targets and Dependencies

Syntax

- Case is significant
- Statement elements (“tokens”) are separated by whitespace
- Every statement ends with a semicolon

Syntax Eye Test

```
X = foo.c ;

x = bar.c ;

X = foo.c;

X foo.c ;

X=foo.c ;
```

Literals

```
X = foo.c ;

foo.c = X ;

X = "this ; and this" ;
```

PERFORCE

Variables

```
X = a b 1 "2 3" ;

"My dog has fleas!" = yes ;

My dog has fleas! = yes ;
```

Variable Expansion

```
X = This is a message ;
Echo $(X) ;

X = Hello ;
$(X) = Bye ;
```

Variable Expansion

```
X = A B C ;

$(X) = Hi there ;

Echo $(X[2]) ;

Echo $(B) ;

Echo $($(X)) ;
```

Adding List Elements

```
X = A B C ;

X = $(X) $(X) ;

X += $(X) ;
```

Variable Expansion Products

```
X = A B C ;

Y = E F ;

Echo $(X) $(Y) ;
   A B C E F

Echo $(X)$(Y) ;
   AE BE CE AF BF CF
```

PERFORCE

Variable Expansion Products

```
X = A B C ;
Y = test_$(X).result ;
Echo $(Y) ;
```

```
test_A.result
test_B.result
test_C.result
```

Variable Expansion Products

```
X = A B C D E F G H ;
Selected = 3 7 8 ;
Echo $(X[$(Selected)]) ;
```

```
  C G H
```

Variable Expansion Products

```
X = Bob Sue Pat ;
Echo "Hello $(X)!" ;
 Hello Bob! Hello Sue! Hello Pat!
Echo Hello $(X)! ;
 Hello Bob! Sue! Pat!
```

Variable Expansion Products

```
X = Bob Sue Pat ;
Echo Hello $(X)$(Y) ;
 Hello
Y = "" "" ;
Echo Hello $(X)$(Y) ;
 Hello Bob Sue Pat Bob Sue Pat
```

Variable Expansion Modifiers

```
X = This is ; Y = A TEST ;
Echo $(X:U) $(Y:L) ;
 THIS IS a test
X = foo.c ; Y = $(X:S=.obj) ;
Echo $(Y) ;
 foo.obj
```

Variable Expansion Modifiers

```
X = foo.c bar.c ola.c ;
Y = .c .obj .exe .dll ;
Echo $(X[2]:S=$(Y):U) ;
 BAR.C BAR.OBJ BAR.EXE BAR.DLL
```

Variable Expansion

- Occurs during parsing phase
- Default value is empty list

Rules & Actions

- "rule" ~= procedure
  - run during parsing phase
  - written in Jam language syntax
- "action" ~= shell script
  - run during update phase
  - written in OS shell command syntax

Defining A Rule

```
rule MyRule
{
  Echo First arg is $(1) ;
  Echo Second arg is $(2) ;
  Echo Third arg is $(3) ;
}
```

Invoking A Rule

```
MyRule a : b c : d e f ;
    First arg is a
    Second arg is b c
    Third arg is d e f
```

Defining An Action

```
actions MyAction
{
  touch $(1)
  cat $(2) >> $(1) ;
}
```

Invoking An Action

```
MyAction ola : foo bar ;
   touch ola
   cat foo bar >> ola
```

More about Actions...

- Actions assume only two arguments
- Arguments are assumed to be buildable "targets"
- Jam variables and variable modifiers can be used in action body

Targets and Dependencies

- A “target” can be:
  - a filesystem object
  - a symbol
- Targets can be given on Jam command line
- Default target is “all” (a symbolic target)

```
rule MyRule {
   Touchfile $(1) ;
}
actions Touchfile {
   touch $(1)
}
MyRule test.output1 ;
MyRule test.output2 ;
MyRule test.output3 ;
```

```
jam -ftest
   don't know how to make all…
   found 1 target(s)…
   …can't find 1 target(s)…

jam -ftest test.output2
   found 1 target(s)…
   updating 1 target(s)…
   Touchfile test.output2
   …updated 1 target(s)…
```

PERFORCE

```
rule MyRule {
   Touchfile $(1) ;
   Depends all : $(1) ;
}
actions Touchfile {
   touch $(1)
}
MyRule test.output1 ;
MyRule test.output2 ;
MyRule test.output3 ;
```

```
jam -ftest
   …found 4 target(s)…
   …updating 2 target(s)…
   Touchfile test.output1
   Touchfile test.output3
   …updated 2 target(s)…
```

```
rule MyRule {
   Touchfile $(1) ;
   Depends all : $(1) ;
}
actions Touchfile {
   touch $(1)
}
MyRule test.output1
       test.output2
       test.output3 ;
```

```
jam -ftest -a test.output2
   …found 1 target(s)…
   …updating 1 target(s)…
   warning: using independent target
     test.output1
   warning: using independent target
     test.output3
   Touchfile test.output1 test.output2
     test.output3
   …updated 1 target(s)…
```

Implicitly Invoked Actions

```
rule MyRule {
  Depends all : $(1) ;
}
actions MyRule {
  p4 info > $(1)
}
MyRule info.output1 ;
```

Target-specific Variables

```
X on foo = A B C ;
X on bar = 1 2 3 ;
```

```
rule MyRule {
   CMD on $(1) = $(2) ;
   PORT on $(1) = $(3) ;
   Depends all : $(1) ;
   MyTest $(1) ;
}
actions MyTest {
   p4 -p$(PORT) $(CMD) > $(1)
}
MyRule test1.output : info ;
MyRule test2.output : info : mars:1666 ;
MyRule test3.output : users : mars:1666 ;
```

```
…found 4 target(s)…
…updating 3 target(s)…
MyTest test1.output
  p4 info > test1.output
MyTest test2.output
  p4 -pmars:1666 info > test2.output
MyTest test3.output
  p4 -pmars:1666 users > test2.output
…updated 3 target(s)…
```

Working Example: A Test Driver Written in Jam

- Simple command tester
- Capturing failed commands
- Comparing canonical results
- Capturing canonical results
- Removing old results and canons
- Writing portable actions

A simple command tester

```
rule Test {
    local f = $(1:S=.out) ;
    Depends all : $(f) ;
    RunTest $(f) ;
    CMD on $(f) = $(1) ;
}

actions RunTest {
    p4 $(CMD) > $(1)
}
```

```
Test info ;
Test users ;
Test clients ;
```

Capturing failed commands

```
rule Test {
  local f = $(1:S=.out) ;
  Depends all : $(f) ;
  RunTest $(f) ;
  CMD on $(f) = $(1) ;
}
actions ignore RunTest {
  p4 $(CMD) > $(1) 2>&1
}
```

```
Test info ;
Test users ;
Test clients ;
```

Comparing canonical results

```
rule Test {
  local f = $(1:S=.out) ;
  CMD on $(f) = $(1) ;
  local canon = $(1:S=.canon) ;
  local match = $(1:S=.match) ;
  Depends all : $(match) ;
  Depends $(match) : $(f) ;
  Depends $(f) : $(canon) ;
  RunTest $(f) ;
  DiffResults $(match) : $(f) $(canon) ;
}
```

```
actions ignore RunTest {
  p4 $(CMD) > $(1) 2>&1
}
actions DiffResults {
  diff $(2) > $(1) 2>&1
}

Test info ;
Test users ;
Test clients ;
```

Capturing canonical results

```
jam
jam -sCAPTURE=1 ...
```

```
rule Test {
    if $(CAPTURE) {
        CaptureCanon $(1) ;
    }
    else {
        DoTest $(1) ;
    }
}
```

```
rule CaptureCanon {
  local canon  = $(1:S=.canon) ;
  local result = $(1:S=.out) ;
  CMD on $(result) = $(1) ;
  Depends all : $(canon) ;
  Depends $(canon) : $(result) ;
  RunTest $(result) ;
  CopyResult $(canon) : $(result) ;
}
actions CopyResult {
  cp $(>) $(<)
}
```

```
rule DoTest {
    local f = $(1:S=.out) ;
    CMD on $(f) = $(1) ;
    local canon = $(1:S=.canon) ;
    local match = $(1:S=.match) ;
    Depends all : $(match) ;
    Depends $(match) : $(f) ;
    Depends $(f) : $(canon) ;
    RunTest $(f) $(match) ;
    DiffResults $(match) : $(f) $(canon) ;
}
```

```
actions ignore RunTest {
    p4 $(CMD) > $(1) 2>&1
}
actions DiffResults {
    diff $(2) > $(1) 2>&1
}
Test info ;
Test clients ;
Test users ;
```

Removing old results and canons

```
jam clean
jam -sCAPTURE=1 clean
```

```
rule Test {
    if $(CAPTURE) {
        CaptureCanon $(1) ;
    }
    else {
        DoTest $(1) ;
    }
}
```

```
rule CaptureCanon {
  local canon  = $(1:S=.canon) ;
  local result = $(1:S=.out) ;
  CMD on $(result) = $(1) ;
  Depends all : $(canon) ;
  Depends $(canon) : $(result) ;
  RunTest $(result) ;
  CopyResult $(canon) : $(result) ;
  Clean clean : $(canon) ;
}
actions CopyResult {
  cp $(>) $(<)
}
```

```
rule DoTest
{
  local f = $(1:S=.out) ;
  CMD on $(f) = $(1) ;

  local canon = $(1:S=.canon) ;
  local match = $(1:S=.match) ;

  Depends all : $(match) ;
  Depends $(match) : $(f) ;
  Depends $(f) : $(canon) ;

  RunTest $(f) $(match) ;
  DiffResults $(match) : $(f) $(canon) ;
  Clean clean : $(f) $(match) ;
}
```

```
actions
  piecemeal together existing Clean {
  rm $(2)
}
actions ignore RunTest {
   p4 $(CMD) > $(1) 2>&1
}
actions DiffResults {
   diff $(2) > $(1) 2>&1
}
Test info ;
Test clients ;
Test users ;
```

Writing portable actions

```
if $(NT) {
  REMOVE = del/f/q ;
  COPY   = copy ;
}
if $(UNIX) {
  REMOVE = rm ;
  COPY   = cp ;
}

actions CopyResult {
  $(COPY) $(>) $(<)
}
```

For More Information

- www.perforce.com/jam/jam.html
  - Jam/MR - Make(1) Redux
  - Using Jamfiles and Jambase
  - Jambase Reference
- Jambase source file
- jamming@perforce.com