**Name** pvs– display the internal version information of dynamic objects

**Synopsis** `pvs [-Cdlnorsv] [-I` *index-expr*`] [-N` *name*`]` *file*`...`

**Description** The `pvs` utility displays any internal version information contained within an ELF file. Commonly, these files are dynamic executables and shared objects, and possibly relocatable objects. This version information can fall into one of two categories:

- version definitions
- version dependencies

Version *definitions* describe the interfaces that are made available by an ELF file. Each version definition is associated to a set of global symbols provided by the file. Version definitions can be assigned to a file during its creation by the link-editor using the `-M` option and the associated *mapfile* directives. See the for more details.

Version *dependencies* describe the binding requirements of dynamic objects on the version definitions of any shared object dependencies. When a dynamic object is built with a shared object, the link-editor records information within the dynamic object indicating that the shared object is a dependency. This dependency must be satisfied at runtime. If the shared object also contains version *definitions*, then those version definitions that satisfy the global symbol requirements of the dynamic object are also recorded in the dynamic object being created. At process initialization, the runtime linker uses any version *dependencies* as a means of validating the interface requirements of the dynamic objects used to construct the process.

**Options** The following options are supported. If neither the `-d` or `-r` options are specified, both are enabled.

`-C` Demangles C++ symbol names.

`-d` Prints version definition information.

`-I` *index-expr* Qualifies the versions to examine with a specific version index or index range. For example, the version with index 3 in an object can be displayed using:

```
example% pvs -I 3 filename
```

An *index-expr* can be a single non-negative integer value that specifies a specific version, as shown in the previous example. Alternatively, an *index-expr* can consist of two such values separated by a colon (:), indicating a range of versions. The following example displays the versions 3, 4, and 5 in a file:

```
example% pvs -I 3:5 filename
```

When specifying an index range, the second value can be omitted to indicate the final item in the file. For example, the following statement lists all versions from the tenth to the end:

```
example% pvs -I 10: filename
```

See Matching Options for additional information about the matching options (`-I`, `-N`).

`-l` Prints any symbols that have been reduced from global to local binding due to versioning. By convention, these symbol entries are located in the *.symtab* section, and fall between the *FILE* symbol representing the output file, and the *FILE* symbol representing the first input file used to generate the output file. These reduced symbol entries are assigned the fabricated version definition `_LOCAL_`. No reduced symbols will be printed if the file has been stripped (see `strip`(1)), or if the symbol entry convention cannot be determined.

Use of the `-l` option implicitly enables the `-s` option

`-n` Normalizes version definition information. By default, all version definitions within the object are displayed. However, version definitions can inherit other version definitions. Under normalization, only the head of each inheritance list is displayed.

`-N` *name* When used with the `-d` option, `-N` prints only the information for the given version definition *name* and any of its inherited version definitions.

When used with the `-r` option, `-N` prints only the information for the given dependency file *name*. It is possible to qualify a specific version from the dependency file by including the version in parenthesis following the file name:

```
example% pvs -N 'dependency (version)' filename
```

See Matching Options for additional information about the matching options (`-I`, `-N`).

`-o` Creates one-line version definition output. By default, file, version definitions, and any symbol output is indented to ease human inspection. This option prefixes each output line with the file and version definition name and can be more useful for analysis with automated tools.

`-r` Prints version dependency (requirements) information.

`-s` Prints the symbols associated with each version definition. Any data symbols from versions defined by the object are accompanied with the size, in bytes, of the data item.

`-v` Verbose output. Indicates any weak version definitions, and any version definition inheritance. When used with the `-N` and `-d` options, the inheritance of the base version definition is also shown. When used with the `-s` option, the version symbol definition is also shown.

## Operands

The following operands are supported.

*file* The ELF file about which internal version information is displayed.

## Usage

Matching Options

The `-I` and `-N` options are collectively referred to as the *matching options*. These options are used to narrow the range of versions to examine, by index or by name.

Any number and type of matching option can be mixed in a given invocation of `pvs`. In this case, `pvs` displays the superset of all versions matched by any of the matching options used. This feature allows for the selection of complex groupings of items using the most convenient form for specifying each item.

## Examples

**EXAMPLE 1** Displaying version definitions

The following example displays the version definitions of `libelf.so.1`:

```
% pvs -d /lib/libelf.so.1
    libelf.so.1;
    SUNW_1.1
```

**EXAMPLE 2** Creating a one-liner display

A normalized, one-liner display, suitable for creating a *mapfile* version control directive, can be created using the `-n` and `-o` options:

```
% pvs -don /lib/libelf.so.1
/lib/libelf.so.1 -    SUNW_1.1;
```

**EXAMPLE 3** Displaying version requirements

The following example displays the version requirements of `ldd` and `pvs`:

```
% pvs -r /usr/bin/ldd /usr/bin/pvs
/usr/bin/ldd:
    libelf.so.1 (SUNW_1.1);
    libc.so.1 (SUNW_1.1);
/usr/bin/pvs:
    libelf.so.1 (SUNW_1.1);
    libc.so.1 (SUNW_1.1);
```

**EXAMPLE 4** Determining a dependency symbol version

The following example displays the shared object from which the `ldd` command expects to find the printf function at runtime, as well as the version it belongs to:

**EXAMPLE 4** Determining a dependency symbol version *(Continued)*

```
% pvs -ors /usr/bin/ldd | grep ' printf'
/usr/bin/ldd -  libc.so.1 (SYSVABI_1.3): printf;
```

**EXAMPLE 5** Determine all dependency symbols from a specific version

The `-N` option can be used to obtain a list of all the symbols from a dependency that belong to a specific version. To determine the symbols that `ldd` will find from version `SYSVABI_1.3` of `libc.so.1`:

```
% pvs -s -N 'libc.so.1 (SYSVABI_1.3)' /usr/bin/ldd

       libc.so.1 (SYSVABI_1.3):
               _exit;
               strstr;
               printf;
               __fpstart;
               strncmp;
               lseek;
               strcmp;
               getopt;
               execl;
               close;
               fflush;
               wait;
               strerror;
               putenv;
               sprintf;
               getenv;
               open;
               perror;
               fork;
               strlen;
               geteuid;
               access;
               setlocale;
               atexit;
               fprintf;
               exit;
               read;
               malloc;
```

Note that the specific list of symbols used by `ldd` may change between Solaris releases.

**EXAMPLE 6** Display base defined version by index

By convention, the base global version defined by an object has the name of the object. For example, the base version of `pvs` is named `'pvs'`. The base version of any object is always version index 1. Therefore, the `-I` option can be used to display the base version of any object without having to specify its name:

```
% pvs -v -I 1 /usr/bin/pvs
       pvs [BASE];
```

**Exit Status** If the requested version information is not found, a non-zero value is returned. Otherwise, a `0` value is returned.

Version information is determined not found when any of the following is true:

- the `-d` option is specified and no version definitions are found.
- the `-r` option is specified and no version requirements are found.
- neither the `-d` nor `-r` option is specified and no version definitions or version requirements are found.

**Attributes** See `attributes`(5) for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|---|---|
| Availability | SUNWtoo |

**See Also** `elfdump`(1), `ld`(1), `ldd`(1), `strip`(1), `elf`(3ELF), `attributes`(5)