Summapack 2026

El `.summa` como fuente única de los paquetes:
las fuentes y los paquetes de cada máquina, desde el proyecto original

Diego Saravia
Facultad de Ciencias Exactas – Universidad Nacional de Salta
INENCO (UNSa – CONICET)
dsa@unsa.edu.ar

17 de noviembre de 2026

Resumen

Summapack es el conjunto de herramientas con que se compilan, empaquetan, publican e instalan los programas de Siru y Simusol. Todo se hace con una sola orden, `sp`: lo que `sp` usa por debajo es interno. Cada proyecto lleva un archivo `.summa` y un `MANIFIESTO`, que son lo único que se edita; de ellos `sp` genera la capa de autotools, los archivos de Debian y el `.spec` de rpm, las fuentes publicables en `tar.gz` y los paquetes en el formato de cada máquina. Al servidor van nuestras fuentes, y el usuario las construye con `sp`, que resuelve las dependencias sin repositorio. Summapack es además la base común de los proyectos: incluye la biblioteca C `mem_util`, y es el gestor de paquetes de Ututo 2027. Se retoman las ideas del Manifiesto Anti-distros (2005), de Rosetta, Babel y Packlets, y del artículo de Sumapack de las JSL 2011, y se indica qué parte de aquellas propuestas está hoy en funcionamiento y qué falta. Este texto refleja el `changelog` del proyecto: el apéndice A dice dónde está contada cada entrada.

Índice

1. Guía rápida **1**
1.1. Publicar una versión nueva . . . 1
1.2. Instalar en otra máquina . . . 1
1.3. El acceso al servidor . . . 2

2. Fundamentos: el Manifiesto Anti-distros y Rosetta **2**
2.1. El Manifiesto Anti-distros . . . 2
2.2. Rosetta, Babel y Packlets . . . 2
2.3. El artículo de 2011 . . . 3

3. Qué es Summapack hoy **3**
3.1. La biblioteca de base: mem_util . . . 3

4. Qué se publica y cómo se instala **4**

5. Las tres capas **6**

6. Plan de instalación y pruebas de sp **6**
6.1. Qué recibe cada usuario . . . 7
6.2. Orden de trabajo y aceptación de cada etapa . . . 7
6.3. Matriz y promoción de cambios . . . 9
6.4. Ejecutar Siru y Calcula desde el árbol . . . 9

7. El archivo .summa **10**
7.1. Campos . . . 11
7.2. Un .summa, varios paquetes . . . 11
7.3. summa.sh . . . 12

8. Armar los paquetes: sp paquete **12**

9. Binarios por distribución: sp binarios **13**

10. Publicar: sp subir y sp estable **14**
10.1. Qué publica cada tipo y cómo probar la instalación por Internet . . . 14

11. Instalar con sp **17**

12. Ututo 2027 **18**

13. La capacidad Ajolote: un requisito de los paquetes modernos **19**
13.1. El paralelo con DKMS . . . 19
13.2. Dónde está puesto (5 de noviembre de 2026) . . . 19
13.3. Qué hace falta en el paquete . . . 20

14. Dónde van los manuales, los lanzadores y los iconos **20**
14.1. Los manuales: `usr/share/doc/simusol/PAQUETE` . . . 20
14.2. La documentación se instala hecha . . . 21
14.3. Los lanzadores y los iconos . . . 21

15. Publicar documentos: subir-doc **22**
15.1. Uso y archivos . . . 22
15.2. Configuración y requisitos . . . 22
15.3. Actualizaciones, respaldos y fallos . . . 23

16. Páginas web: herramientas y lenguajes **23**
16.1. Orden y referencia de sincronización . . . 24

17. Por dentro: referencia de los componentes **24**
17.1. Elegir una operación . . . 25
17.2. summapack y distro . . . 25
17.3. sp y el motor interno de descarga . . . 25
17.4. instalar-tgz . . . 27
17.5. preparar, fuentesexternas y dependencias . . . 27
17.6. packinstall y diccionariodepaquetes . . . 27
17.7. summa.sh y la receta de sp paquete . . . 28
17.8. Publicadores y configuración . . . 28
17.9. icon_install, findbinlib y templatesumma . . . 29
17.10 distros, mem_util, configuración y pruebas . . . 29
17.11 makefile.lib y ubicación de sus archivos . . . 30
17.12 Las demás herramientas de bin/ . . . 31

18. Proyectos **32**

19. Flujo de trabajo **32**

20. Lo que falta: hoja de ruta **33**

A. Historia de los cambios **33**

B. bin/ de summapack: relevamiento (07/10/2026) **34**
B.1. Índice vigente de todos los componentes . . . 34
B.2. Actualización de distro, versión 4 . . . 37
B.3. Lo que sale de esto . . . 41
B.4. Hecho (07/10/2026, pasos 1 y 2) . . . 42
B.5. Hecho (07/10/2026, pasos 3 y 4) . . . 42

C. Entornos de construcción y prueba **43**
C.1. Protección durante las pruebas . . . 43

D. siru/summapack/distros: compilar en otras distribuciones **44**
D.1. Uso . . . 44
D.1.1. Diez entornos mínimos y `sp` . . . 45
D.2. Que hay . . . 45
D.3. Como funciona . . . 46
D.4. Pruebas locales actuales (12/10/2026) . . . 47
D.5. Lo que aparecio al compilar alli . . . 47
D.6. Probado (07/10/2026) . . . 47

E. Agregado al Manifiesto Anti-distros: la capacidad Ajolote **47**
E.1. El texto . . . 48

F. Formatos de paquete **48**
F.1. Pasos comunes . . . 48
F.2. Herramientas y estado . . . 49
F.3. Desinstalar instalaciones directas . . . 49
F.4. Imagen de Linux From Scratch . . . 50
F.5. Imagen de Trisquel . . . 50

G. Instalar summapack sin red y sin herramientas de empaquetado **50**
G.1. Lo que hay que entender primero . . . 51
G.2. El recorrido: `sp` sin argumentos, desde el arbol . . . 51
G.3. El ultimo recurso: `hacer-pkg-minimo` . . . 51
G.3.1. Que se instala . . . 52
G.3.2. Las dependencias no se declaran . . . 52
G.3.3. La version lleva `~minimo` . . . 52
G.4. Sin arbol: el `sp` suelto con el paquete adentro . . . 52
G.4.1. Como esta hecho ese `sp` . . . 53
G.4.2. Como instala sin red . . . 53
G.5. Como se escribe cada formato a mano . . . 54
G.5.1. El `.deb` . . . 54
G.5.2. El `.rpm` . . . 54
G.5.3. El `.tgz` . . . 54
G.6. mem_util compilada a mano . . . 55
G.7. Que se midio, y con que . . . 55
G.8. Que NO se midio . . . 56
G.9. Los archivos . . . 57

H. Instalar paquetes de Simusol sin summapack **57**
H.1. Como esta organizado el servidor . . . 57
H.2. Paso a paso . . . 57
H.3. Ejemplo: Tabular en Ubuntu 24.04 amd64 . . . 58

I. Inventario de proyectos con Summa **58**
I.1. Verificación del 14 de noviembre de 2026 . . . 59

J. Evaluación de migración de comandos a Bash **60**

K. Sincronizar Markdown y CMSimple XH **61**
K.1. Herramientas, nombres y lenguajes . . . 62
K.2. Decisión automática . . . 62
K.3. Orden completo de una sincronización . . . 63
K.3.1. Qué significa la base común . . . 63
K.3.2. Implementado y pendiente . . . 64
K.4. Requisitos y configuración . . . 64
K.5. Tabla central . . . 64
K.6. Uso: un solo comando . . . 65
K.7. Respaldos y garantías . . . 65
K.8. Generadores existentes y alcance . . . 66
K.9. Pruebas y salidas . . . 66

L. SummaPack **67**

M. SummaPack: páginas para CMSimple / TinyMCE **75**
M.1. Generar y revisar . . . 75
M.2. Publicar con TinyMCE . . . 75
M.3. Fuentes y conservación . . . 76

N. Esquema: de nuestras fuentes al usuario, todo con `sp` **76**
N.1. El recorrido . . . 76
N.2. Las tres capas . . . 77
N.2.1. La documentación se instala hecha . . . 77
N.3. Qué hay en el servidor . . . 78
N.4. Las fuentes van por formato; los binarios, por distribución . . . 78
N.5. Lo que hacemos nosotros . . . 79
N.5.1. 1. Mantener las fuentes . . . 79
N.5.2. 2. Armar . . . 79
N.5.3. 3. Publicar . . . 79
N.5.4. Orden de un lanzamiento completo . . . 80
N.6. Lo que hace el usuario . . . 80
N.6.1. Sin nada instalado . . . 80
N.6.2. Con `sp` instalado . . . 80
N.6.3. Desde el árbol de svn, sin internet . . . 81
N.7. Lo que hace un empaquetador de otra distro . . . 81
N.8. Lo que todavía falta . . . 81

Ñ. Paquetes: del .summa al servidor de descargas **81**
Ñ.1. Guia rapida . . . 81
Ñ.1.1. Publicar una version nueva . . . 82
Ñ.1.2. Instalar en otra maquina . . . 82
Ñ.1.3. Preparar el acceso al servidor (hecho el 07/10/2026) . . . 82

Ñ.2. Otras distribuciones: el problema . . . 83
Ñ.3. De donde viene . . . 84
Ñ.4. Summapack, la base comun . . . 84
Ñ.5. El .summa . . . 85
Ñ.6. hacer-pkg . . . 86
Ñ.7. Interfaz de construcción e instalación: objetivo y estado . . . 86
Ñ.7.1. `make distcheck` es la prueba de todo esto (08/10/2026) . . . 88
Ñ.8. El servidor de descargas . . . 88
Ñ.9. sp subir (18/10/2026) . . . 89
Ñ.9.1. El arranque (bootstrap) . . . 90
Ñ.10.subir-paquetes . . . 90
Ñ.11.bajar-paquetes e instalar-tgz . . . 91
Ñ.12.Donde estan las herramientas . . . 92
Ñ.13.Proyectos con .summa (07/10/2026) . . . 92
Ñ.14.Publicar los documentos . . . 93
Ñ.15.Desde las fuentes: construir localmente, instalar el paquete . . . 93
Ñ.16.Catálogo histórico de fuentes externas (etapa 2, 07/10/2026) . . . 95
Ñ.17.Que las fuentes de cada proyecto se preparen solas (etapa 3) . . . 95
Ñ.18.Un solo juego de paquetes: base Ubuntu 22.04 y "comunes" (08/10/2026) . . . 95
Ñ.19.Producción incremental de todos los paquetes . . . 97
Ñ.20.Requisito: lo instalado desde fuentes se puede desinstalar (18/10/2026) . . . 98
Ñ.21.Prueba en los diez entornos (19/10/2026) . . . 98
Ñ.22.Los .rpm desde el .summa: summa.sh rpm (19/10/2026) . . . 99
Ñ.23.Los binarios de las bases viejas: armar-paquetes y probar-paquetes (19/10/2026) 99
Ñ.24.Desde el arbol de svn: --arbol (19/10/2026) . . . 100
Ñ.25.MANIFIESTO: las fuentes de cada proyecto (20/10/2026) . . . 100
Ñ.26.Versiones: AÑO.MAYOR.MENOR.rREV-N (20/10/2026) . . . 101
Ñ.27.Que decide el .summa (20/10/2026) . . . 102
Ñ.28.Referencia, seguridad y límites . . . 103

O. makefile.lib: catálogo **104**
O.1. 1. Cómo se usa: el proyecto mínimo . . . 104
O.2. 2. El `.summa` y los archivos de configuración . . . 105
O.3. 3. Variables . . . 106
O.4. 4. Funciones (`$(call nombre,arg1,arg2,...)`) . . . 107
O.4.1. Instalar . . . 107
O.4.2. Bajar, buscar, quitar . . . 107
O.5. 5. Objetivos . . . 108
O.6. 6. Cómo se prueba . . . 109
O.7. 7. Problemas: los arreglados y los que quedan . . . 109
O.8. 8. ¿Y para empaquetar calcula y siru? (paso 6, 07/10/2026) . . . 110
O.9. 9. Resumen del plan del 07/10/2026 . . . 111

P. Contrato de `sp` **111**
P.1. Rosetta, Babel y packlets . . . 111
P.2. Interfaz de packlets . . . 112
P.3. Operaciones . . . 112
P.4. Traducción de paquetes . . . 113
P.5. Límites . . . 113

Q. sp: interfaz de paquetes de SummaPack **113**
Q.1. Instalación y ayuda . . . 113
Q.2. Sintaxis . . . 114
Q.2.1. Paquetes ya instalados y compatibilidad con `packinstall` . . . 114
Q.3. Acciones . . . 114
Q.4. Objetos de instalación . . . 116
Q.5. Distribuciones y traducción . . . 116
Q.6. Privilegios, salida y códigos . . . 117
Q.7. Ejemplos . . . 117
Q.8. Desarrollo y pruebas . . . 118

R. Arreglar las instrucciones de instalación para que digan cómo usar `sp` **118**
R.1. Texto canónico que debe aparecer en todos lados . . . 118
R.2. Qué cambiar, dónde . . . 119
R.3. Regla para no volver a divergir . . . 120

S. Incidencia resuelta: Slex exigía un mem_util más nuevo que el publicado **120**
S.1. El problema . . . 120
S.2. Qué correr, en orden . . . 120
S.3. Verificación . . . 121
S.4. Para que no se repita . . . 121

1. Guía rápida

1.1. Publicar una versión nueva

Con todo comiteado, parado en el árbol o dando `LUGAR/NOMBRE`:

```
sp manifiesto LUGAR/NOMBRE          # el MANIFIESTO al dia con el arbol
sp paquete LUGAR/NOMBRE             # fuentes (tar.gz), .summa con version y paquetes
sp binarios LUGAR/NOMBRE ututo2027  # binarios en el Podman de cada distribucion
sp subir --prueba LUGAR/NOMBRE      # que subiria, sin subir
sp subir LUGAR/NOMBRE               # sube, como inestable
sp estable LUGAR/NOMBRE             # lo inestable pasa a estable
sp subir --bootstrap LUGAR/summapack  # si cambio summapack: el sp autonomo
```

Comitear no autoriza a subir: cada subida la decide el autor. El orden de un lanzamiento completo y lo que deja cada paso están en la sección 10.

1.2. Instalar en otra máquina

La guía pública es `https://www.simusol.org/downloads/0-README-como-instalar`. En una máquina sin Summapack:

```
curl -fsSLO https://www.simusol.org/downloads/sp
sh sp --para fuentes-paquete simusol
```

El `sp` autónomo instala Summapack y después el proyecto con todas sus dependencias. Con `sp` instalado:

```
sp --para fuentes install simusol            # make install directo
sp --para fuentes --prefijo ~/.local install simusol   # en el home, sin root
sp --para fuentes-paquete install simusol    # preferida: compilar aca e instalar como
    paquete
sp install simusol                           # el binario de esta distribucion
sp remove simusol
```

Las formas de instalar se explican en la sección 11.

1.3. El acceso al servidor

En simusol.org, www-data no entra por ssh (intentarlo hace que fail2ban bloquee la IP). Se publica con un usuario propio del grupo www-data que solo escribe en `downloads/`. El destino lo dice `SUBIR_A` en el `.summa`; `sp subir` comprueba antes que se entre por ssh sin contraseña y, si no, no sube nada. Cómo se creó el usuario está en el apéndice Ñ.

2. Fundamentos: el Manifiesto Anti-distros y Rosetta

2.1. El Manifiesto Anti-distros

En el *Manifiesto Anti-distros* [1] se planteaba que el empaquetado debía ser un trabajo vinculado estrechamente al desarrollo de cada programa y no a cada distribución: que los `Makefile` de cada proyecto pudieran producir los paquetes necesarios —"deb, rpm, tgz"— y que la diferencia entre un deb y un rpm fuera "solo de formato, como puede ser la diferencia entre un gz y un bz2". La crítica era a las distribuciones como burocracias: formatos, repositorios y herramientas incompatibles entre sí, que desplazan el poder de quienes desarrollan los programas a quienes los empaquetan.

Summapack hoy es la respuesta práctica a esa crítica:

- **El empaquetado es del proyecto.** Lo que dice cómo se arma, qué instala y de qué depende está en el `.summa` del proyecto, no en una distribución (sección 7).
- **deb, rpm, pacman y tgz son formatos.** `sp paquete` arma el que corresponda a la máquina desde las mismas fuentes y la misma instalación (sección 8).
- **Sin repositorio.** Al servidor van nuestras fuentes; el usuario las construye con `sp`, que resuelve las dependencias de cada distribución (sección 4).
- **Cada paquete puede volver a existir.** Es la capacidad Ajolote, el requisito que se propone agregar al manifiesto (sección 13, y el apéndice E).

2.2. Rosetta, Babel y Packlets

La página *Rosetta, Babel y Packlets* [2] describe el proceso de distribución del software libre —control de cambios, paquetes originales, información de empaquetado, repositorios, bases de datos locales— y propone tres piezas para atravesar la fragmentación de las distribuciones. Hoy son estas:

Pieza	Hoy
Rosetta: el nombre de cada paquete en cada distribución	`diccionariodepaquetes`, una línea `NOMBRE_DEBIAN:DISTRO:NOMBRE_ALLA` por paquete y distribución. Lo usan `sp` al instalar dependencias, el `.spec` de rpm y el paquete de pacman.
Babel: instalar con la herramienta de cada sistema	`sp`: una sola interfaz (`install`, `remove`, `search`, ...) que llama a apt, dnf, zypper, pacman o al registro propio de tgz.
Packlets: el adaptador de cada gestor	`sp/packlets/`: `apt`, `dnf`, `zypper`, `pacman`, `portage`, `urpmi`, `tgz`, `podman` y `summapack`, uno por archivo.

2.3. El artículo de 2011

El artículo de Sumapack [3] describió el primer sistema construido con esas ideas. Entre sus componentes: Rosetta y Babel, AutoSumma (extensión de las autotools para generar paquetes deb, rpm, txz o ebuild, donde "los archivos que portan información específica del paquete en cada tipo: `.spec`, `.lsm`, `control`, `.ebuild`, se generan desde un archivo común"), Uinstall, Upack, Upackname, Ubuild y un Bootstrap. Este trabajo vuelve a aquel "archivo común": el `.summa`. Los textos del manifiesto, de Rosetta y del artículo están en `doc/`.

3. Qué es Summapack hoy

Un solo proyecto (`siru/summapack`), un solo `summapack.summa`, y tres paquetes:

Paquete	Contenido	Arquitectura
`summapack`	`sp` y sus packlets, los generadores del `.summa` (`summa.sh`, `summa-debian.sh`, `summa-automake.sh`, `summa-rpm.sh`), la receta de `sp paquete`, `diccionariodepaquetes`, `makefile.lib` y las herramientas de `bin/`	all
`libmem-util0`	la biblioteca C `mem_util`	any
`libmem-util-dev`	sus encabezados, el `.a`, el enlace de desarrollo y `mem_util.pc`	any

`summapack` depende de `libmem-util-dev`, y de `curl`, con el que `sp` baja todo (también de `file://`): alcanza con pedir summapack. Es la capa básica de los proyectos, así que está escrita en shell: Bash y herramientas elementales, sin Python ni Perl para lo esencial.

3.1. La biblioteca de base: mem_util

`mem_util` reúne lo que todos los programas de Siru y Simusol necesitaban y cada uno reimplementaba: crecimiento de vectores en memoria, apertura y búsqueda de archivos, conversión entre rutas relativas y absolutas, y fechas. Desde octubre de 2026 `datetime()` devuelve segundos desde 1970 (epoch), respetando la zona local o la indicada en el dato. Vive en `summapack/mem_util` y se empaqueta con summapack.

4. Qué se publica y cómo se instala

Decisión del autor del 16 de noviembre de 2026. El usuario típico no instala binarios: baja nuestras fuentes y `sp` las construye en su máquina. Quien usa sceptre ya compila Fortran en cada simulación, así que las herramientas para compilar tienen que estar igual.

Qué	Para quién	Lo arma
el `sp` autónomo, con summapack adentro, y la guía `0-README-como-instalar`	todos	`sp subir -bootstrap`
`PROYECTO.summa`	`sp`: la versión estable y las dependencias	`sp paquete`
`PROYECTO/NOMBRE_VERSION.tar.gz`	el usuario típico: lo compila `sp`	`sp paquete`
binarios por distribución	quien prefiera no compilar	`sp binarios`
fuentes por formato (`.dsc`, `.src.rpm`)	un empaquetador de otra distribución	`sp fuentes`
la ISO de Ututo 2027	quien	`sp iso ututo2027`

Cada archivo va con su `.md5`. **No hace falta repositorio apt ni dnf**: `sp` resuelve las dependencias. Las de la distribución las instala con el gestor de la máquina, traduciendo los nombres (Rosetta); las nuestras las baja y las construye igual que el proyecto pedido. El `.summa` viaja dentro de las fuentes aunque nadie lo use.

Las fuentes van **por formato** y los binarios **por distribución**: un mismo `.src.rpm` sirve para Fedora y para openSUSE, porque las dependencias que se llaman distinto quedan escritas como alternativas (`(gcc-gfortran or gcc-fortran)`); en cambio, cada distribución compila contra sus propias bibliotecas, y la marca va en el nombre del binario. El recorrido entero, sólo con órdenes de `sp`, está en el apéndice N.

5. Las tres capas

Capa	Qué es	¿Se edita?	¿Se versiona?	¿Viaja?
summapack	el `.summa`, el `MANIFIESTO` y los archivos del proyecto	sí: es lo único	sí	sí
automake	`instalar.am`, `dist.am`, `configure`, `Makefile.in`	nunca: la genera `sp`	no	sí
deb / rpm	`debian/control`, `rules`...; el `.spec`	nunca: la genera `sp`	no, salvo `debian/changelog`	sí

Lo generado no se versiona pero se distribuye: el `tar.gz` publicado trae el `configure` hecho, como cualquier paquete GNU, y la máquina que instala no necesita autoconf, automake ni libtool.

El MANIFIESTO. Enumera las fuentes del proyecto. Si existe `MANIFIESTO.excluidos`, la copia de construcción incorpora los archivos nuevos que cumplan las reglas de inclusión, avisa y los anota en su propio manifiesto. Sin ese archivo de exclusiones se copia la lista declarada. El paquete debe verificar el contenido que entrega.

Las fuentes por formato. `sp fuentes LUGAR/NOMBRE [SALIDA]` da, de nuestras fuentes, las de cada formato de empaquetado:

Archivo	Qué es
`NOMBRE_V.orig.tar.gz`	el programa, con la capa automake y sin `debian/`: también la fuente de `./configure && make`
`NOMBRE_V-N.debian.tar.xz`	la capa deb
`NOMBRE_V-N.dsc`	la ficha de Debian que une los dos
`NOMBRE-V-N.src.rpm`	la capa rpm: el `.spec`, con el mismo tarball como `Source0`

Con ellas, un empaquetador de otra distribución no necesita summapack: `dpkg-source -x` y `dpkg-buildpackage`, o `rpmbuild -rebuild`. `sp subir -tipo distro` las publica (sección 10).

6. Plan de instalación y pruebas de sp

Para trabajar en los proyectos de Simusol se prefiere construir con las fuentes disponibles en la máquina destinataria y registrar el resultado con su gestor nativo. También se admiten

la instalación directa en un prefijo y los binarios publicados. La selección del origen, el tipo de fuente, el entorno de construcción y la forma de instalación son decisiones distintas. Este plan toma como referencia el árbol en r9377. Las etapas siguientes indican trabajo a verificar o desarrollar.

6.1. Qué recibe cada usuario

Fuentes de desarrollo. Archivos propios, `.summa` y `MANIFIESTO`. En SVN se conoce la revisión. También se puede indicar un directorio local sin SVN. Se respeta la versión ya escrita en una entrega; una copia de desarrollo sin revisión puede recibir el sufijo `+local`. Hay que anticipar si el gestor considera esa versión anterior a la instalada.

Fuentes GNU generadas. El tarball con `configure` y `Makefile.in` debe compilarse sin SummaPack ni herramientas para regenerar Autotools. Probar `configure`, `make`, las pruebas del proyecto, instalación con `DESTDIR`, prefijo elegido y desinstalación. Para instalar una entrega GNU vía paquete con `sp`, ver las recetas admitidas más abajo.

Fuentes por formato. El empaquetador Debian recibe el `.dsc` con sus archivos asociados; el de RPM recibe un `.src.rpm`. Cada uno construye con las herramientas de su distribución. En r9377 hay un packlet para un directorio fuente Debian expandido; la entrada normal `install SOURCE` todavía atiende proyectos propios mediante `summapack.sh`. Registrar por separado qué formatos se generan, se reconocen como entrada y se han probado instalando.

Binarios. Quien pide binarios recibe los construidos en el Podman de la base antigua soportada de su familia. Fedora y SUSE tienen RPM propios. Verificar en otra imagen compatible y probar Trisquel y Ututo con sus paquetes correspondientes. Registrar el formato real de Arch y el TGZ genérico por separado. La construcción local usa el entorno local; en Podman se detecta la distribución dentro del contenedor.

```
sp --arbol ~/svn --para fuentes-paquete install SOURCE calcula
sp --arbol /ruta/a/fuentes --para fuentes-paquete install SOURCE calcula
sp --sitio --para fuentes-paquete install SOURCE calcula
sp --arbol ~/svn --para fuentes --prefijo ~/.local install SOURCE calcula
```

Los directorios de los dos primeros ejemplos deben contener los proyectos propios necesarios. En la forma directa hay que probar qué archivos registra el desinstalador para cada prefijo.

6.2. Orden de trabajo y aceptación de cada etapa

1. **Fijar la referencia.** Registrar las órdenes existentes, su código de salida, archivos instalados, propietario y versión. Usar el proyecto de ejemplo y una raíz descartable antes de tocar proyectos reales. Para cada cambio de comportamiento anotar el caso anterior y el esperado. Conservar los nombres de acciones y los registros creados por versiones anteriores.

2. **Acotar las operaciones sobre la máquina.** Probar con caché vigente, vencida y ausente. Consultar y buscar debe funcionar como usuario; instalar un paquete local con dependencias satisfechas debe funcionar sin red. La actualización general es una operación explícita. Pacman requiere tratar la sincronización y actualización conjunta. Compilar como usuario y elevar privilegios al instalar dependencias o el resultado; el registro de un fallo debe poder leerlo quien inició la corrida.

3. **Resolver el pedido antes de compilar.** Conservar alternativas, comparadores y versiones mínimas al traducir dependencias. Comprobar la fuente elegida, arquitectura y versión

candidata. Incluir proyectos con más de un paquete, copias históricas, rutas con espacios y cambios locales. El ensayo debe mostrar la misma selección que la ejecución. Validar antes de construir los reemplazos y degradaciones; impedir eliminaciones ajenas al reemplazo autorizado.

4. **Registrar el progreso.** Aprovechar el cartel por proyecto y la orden de continuación de r9376. Verificar el resultado con el gestor. Probar un lote A, B, C donde B falla: A se conserva, el registro de B queda accesible y la continuación comprende B y C. Diseñar el estado persistente de la corrida y la consulta del conjunto completo antes de documentar nuevas órdenes. En una reanudación, comprobar que siguen coincidiendo fuentes, arquitectura y artefactos guardados.

5. **Completar los packlets por entrada.** Primero el paquete fuente Debian, incluidos sus subpaquetes; luego RPM con sus pruebas separadas en Fedora y SUSE, y las fuentes Autotools. Para Arch y genérico comprobar el constructor y el instalador reales. Reutilizar los traductores, la detección y las herramientas nativas. Al activar un packlet, exigir que se pueda construir e instalar su ejemplo desde una entrega aislada y que el usuario conserve las demás formas.

6. **Verificar instalación y desinstalación.** Para DEB, RPM y Pacman consultar al gestor. Para TGZ y prefijos privados, probar respaldo, colisiones, actualización, interrupción y desinstalador. Dos TGZ que reclaman la misma ruta deben resolver el conflicto antes de escribir; al quitar uno, verificar el contenido y el propietario de los archivos del otro.

7. **Completar la implementación en Bash básico.** Inventariar los auxiliares, incluido `sp-instalados`, que en r9377 utiliza Python. Sustituirlos de uno en uno, manteniendo argumentos, estado de salida y lectura de registros antiguos. Ejecutar las pruebas en el Bash de la base antigua. El requisito se aplica a `sp` y sus auxiliares propios; las pruebas funcionales incluyen proyectos escritos en otros lenguajes.

Avances verificados después de r9377. Las consultas del gestor son de lectura, la construcción desde fuentes vía paquete se ejecuta como usuario y el registro TGZ comprueba los propietarios entre paquetes. Para usar las fuentes específicas de distribución, colocar el archivo y sus componentes en la máquina de la familia correspondiente con las herramientas y dependencias de compilación instaladas:

```
sp --para fuentes-paquete install SOURCE ./proyecto.tar.gz
sp --para fuentes-paquete install SOURCE ./proyecto.dsc
sp --para fuentes-paquete install SOURCE ./proyecto.src.rpm
sp --para fuentes-paquete install SOURCE ./PKGBUILD
```

Una entrega GNU comprimida con gzip, xz o bzip2 puede llevar su receta de instalación. En Debian, Ubuntu o Trisquel, `sp` usa el `debian/` incluido. En Arch usa el `PKGBUILD` de la entrega. El tarball puede contener los archivos directamente o dentro de un directorio principal. El ensayo identifica la receta y muestra qué herramientas usaría. Un tarball con solo `configure` conserva su instalación GNU habitual. Para instalarlo vía paquete se necesita además la receta que declara los subpaquetes y las dependencias. En Fedora y SUSE se usa el `.src.rpm` de la distribución.

Debian también admite el directorio fuente expandido. Los binarios producidos se instalan con APT, DNF, Zypper o Pacman. Pueden existir subpaquetes cuyo nombre difiere del proyecto fuente.

Para el conjunto Siru y Calcula, `sp -arbol DIR -para fuentes-paquete install todos` usa el catálogo `conjunto-siru-calcula`; `sp estado` consulta las versiones instaladas de los paquetes y devuelve estado 1 si falta alguno. DIR debe abarcar ambos grupos de fuentes. Antes de instalar el primer proyecto del lote, se comprueba que estén las fuentes de cada proyecto

pedido. Si dos directorios ofrecen la misma versión máxima, la orden muestra ambos para que se pueda elegir un árbol más específico. Un proyecto instalado directamente en un prefijo privado se consulta mediante `sp list-source`.

El lote local vía paquete registra los pendientes después de cada proyecto en `$XDG_STATE_HOME/summapack/c` (por omisión, bajo `~/.local/state`). Tras un fallo, `sp reanudar` continúa la única corrida pendiente. Si hay varias, se elige con `sp reanudar ID`. Se comprueba el bloqueo de la corrida y se vuelven a leer las fuentes pendientes. Un proyecto interrumpido antes de registrar su éxito puede volver a construirse.

6.3. Matriz y promoción de cambios

- **Cada cambio.** Pruebas rápidas de selección, traducción, modos de lectura, errores, confirmación final y reintentos. Comprobar que incluir un archivo nuevo en la copia de construcción respeta las exclusiones. Verificar tanto el contenido del paquete como el estado de SVN después de la operación.
- **Cada packlet o generador modificado.** Construir en un entorno limpio e instalar en otro, con el árbol desmontado. Probar instalación inicial, actualización, reemplazo, reinstalación y retiro. Ejecutar el mismo caso de fuentes propias con SVN y sin SVN. Controlar el resultado de una versión `+local` sobre una revisión instalada.
- **Antes de publicar.** Usar las bases antiguas de producción y las imágenes destinatarias nuevas. Probar los 17 proyectos de Siru y Calcula y sus 24 paquetes de referencia, incluyendo documentación y corpus. Verificar una cuenta de Calcula, Tabular, Radia, un deck de SCEPTRE que compile y ejecute, y Dia–Simusol–Simplot con una cuenta nueva. Registrar por entorno el artefacto, versión, pruebas y resultado; diferenciar ensayo y ejecución real.
- **Si falla.** Conservar artefacto y registro, detener la promoción de esa etapa y reproducir en una raíz descartable. Versionar cada etapa por separado. Probar volver al paquete anterior en una copia de la instalación y verificar registros y archivos antes de decidir una reversión en una máquina de uso.

6.4. Ejecutar Siru y Calcula desde el árbol

```
sp --ensayo --arbol ~/svn activar todos
sp --arbol ~/svn activar todos
sp estado
sp estado simusol
sp desactivar todos
```

Se puede elegir un solo proyecto en lugar de todos. Por ejemplo, al activar Simusol, `/usr/bin/simusol1.pl` apunta a `bin/simusol1.pl` de su árbol. La consulta `sp estado simusol` enumera cada ruta instalada y su origen.

Mientras un proyecto está activo, sp comprueba el registro antes de instalarlo o quitarlo. Los Makefiles actuales consultan el mismo registro al pedir instalación o desinstalación. Se puede compilar, correr las pruebas y armar paquetes con make install DESTDIR=/ruta/temporal. Para instalar sobre el sistema hay que desactivar primero el proyecto. Una entrega de fuentes generada antes de este cambio conserva sus recetas originales.

Instalá primero los paquetes del proyecto y compilá en el árbol los artefactos que vayas a usar con el mismo prefijo que los paquetes. Por ejemplo, en unidades usá ./configure –prefix=/usr antes de compilar para que la biblioteca encuentre definitions.units en /usr/share/unidades. Si ya se había compilado con otro prefijo, volvé a compilar también los objetos existentes (make -B). El directorio puede ser una copia SVN o una fuente expandida con .summa y MANIFIESTO.

Para todo el conjunto, elegí el directorio que contiene Siru y Calcula. Esta modalidad funciona con APT y con DNF que disponga de versionlock.

La activación usa sudo y vale para todos los usuarios de la máquina, incluso desde el escritorio. Usá un árbol confiable y accesible para ellos. Los scripts y datos se leen directamente del árbol. Para cambiar una biblioteca o un ejecutable compilado hay que reconstruirlo en su ubicación de desarrollo y volver a iniciar el programa que lo usa.

sp cruza el inventario de cada paquete instalado con las correspondencias del archivo enlaces-desarrollo. Enlaza los archivos regulares que tienen contraparte y conserva los enlaces existentes del paquete. Los archivos nuevos que todavía no figuran en el paquete requieren actualizar su inventario mediante una nueva instalación. Para actualizar el paquete o cambiar de árbol, desactivá el proyecto primero.

Antes de sustituir archivos, sp comprueba el conjunto y respalda los originales instalados en /var/lib/summapack/desarrollo. Retiene los paquetes mediante apt-mark hold o dnf versionlock. Al desactivar, restaura los archivos desde el respaldo local aunque se haya movido el árbol, y libera solamente los bloqueos que agregó. Si alguien reemplazó un enlace o cambió la versión instalada, sp informa el conflicto y conserva el respaldo. Una operación interrumpida se recupera al volver a ejecutar activar o desactivar. Si el árbol de SummaPack quedó inaccesible y el enlace a sp dejó de funcionar, ejecutá sudo /var/lib/summapack/desarrollo/desactivar todos.

7. El archivo .summa

Cada proyecto tiene en su directorio un `NOMBRE.summa` y un `MANIFIESTO`. El `.summa` es un archivo de shell y es *la* fuente: si alguien cambia el `.summa`, cambian todos los paquetes. En `debian/` queda únicamente `changelog`, que es la historia y lleva la revisión (el `-38` de `6.0-38`); su versión de arriba tiene que coincidir con `BASEVERSION`.

7.1. Campos

Campo	Qué es
`BASEVERSION`	la versión de arriba (`6.0`)
`SUMMARY`, `DESCRIPTION`	resumen y descripción del paquete principal
`DEPENDS`	sus dependencias (admite `${shlibs:Depends}`)
`MAINTAINER`, `HOMEPAGE`, `SECTION`, `PRIORITY`, `LICENSE`, `STANDARDS`	los datos generales
`BUILDDEPENDS`	lo necesario para compilar
`CONFIGURE`	las opciones de `./configure` (`-with-klu`, ...)
`FUENTES_PREPARAR`	un paso propio que corre en la copia antes de armar las fuentes (generar un `.tab.c` con bison, por ejemplo)
`SUBIR_A`	adonde lo sube `sp subir`; en lo publicado queda la plantilla
`DH`	las opciones de `dh`
`RULES`	el resto de `debian/rules`, lo propio del proyecto
`SOURCE_EXTRA`	otros campos del paquete fuente
`PACKAGES`	los paquetes que salen; el primero es el principal
P`_SUMMARY`, *P*`_DESCRIPTION`, *P*`_DEPENDS`	los de cada paquete *P* que no es el principal
P`_ARCH`, *P*`_EXTRA`	arquitectura y otros campos de control (Section, Multi-Arch, Suggests, ...) de cada paquete
`ARCHIVOS`, `ARCHIVO_`*ruta*	los demás archivos de `debian/` (copyright, *P*`.install`, *P*`.docs`, postinst, ...), cada uno entero; la ruta puede tener subdirectorios
`EJECUTABLES`	cuáles de ellos van con permiso de ejecución

En los nombres de variable, *P* es el nombre del paquete con `_` en lugar de `-`, `.` y `+` (por ejemplo, `libmem_util_dev_SUMMARY`). Los valores de varias líneas se escriben como documento en línea:

```
BASEVERSION="2025.01.01"
SUMMARY="C++ Mathematical Expression Toolkit Library (cabecera)"
DESCRIPTION=$(cat << 'FIN_SUMMA'
exprtk evalua expresiones matematicas escritas como texto: variables,
funciones, condicionales y las operaciones habituales, ...
FIN_SUMMA
)
BUILDDEPENDS="debhelper-compat (= 13)"
DH="--makefilename=Makefile.simple"
PACKAGES="libexprtk-dev"
libexprtk_dev_ARCH="all"
```

7.2. Un .summa, varios paquetes

Un proyecto puede dar varios paquetes. Summapack da tres; tabular2 da `libtabular2-0`, `libtabular2-dev` y `tabular`; slex da `libslex0`, `libslex-dev` y `d2datdinc`. Todo se instala en `debian/tmp` y cada `ARCHIVO_`*P*`_install` elige lo suyo:

```
PACKAGES="summapack libmem-util0 libmem-util-dev"
summapack_ARCH="all"
libmem_util0_ARCH="any"
libmem_util_dev_DEPENDS="libmem-util0 (= \${binary:Version}), \${misc:Depends}"
ARCHIVO_summapack_install=$(cat << 'FIN_SUMMA'
```

```
debian/tmp/etc
usr/bin/*
usr/share/summapack
FIN_SUMMA
)
```

Si una ruta existe también en el árbol fuente (como `etc` en summapack), hay que escribirla `debian/tmp/etc`: `dh_install` busca primero en el árbol fuente.

La lista de instalación (`ARCHIVO_P_install`, o `ARCHIVO_install` si hay un solo paquete) dice, en renglones `ORIGEN DESTINO [NOMBRE]`, qué se instala y adónde; de ella salen el `make install` (`instalar.am`) y cada paquete. `!RUTA` no se copia, y `enlace DESTINO OBJETIVO` hace un enlace.

7.3. summa.sh

Todo lo que lee o genera el `.summa` es Bash, sin Python (13 al 15 de noviembre de 2026: reemplazó a `summa.py`). Lo usa `sp`; a mano sólo hace falta para consultar:

Orden	Qué hace
`summa.sh campo DIR CAMPO`	el valor de un campo
`summa.sh dependencias DIR`	los nombres de `BUILDDEPENDS`
`summa.sh debian DIR`	genera `DIR/debian/` (`summa-debian.sh`)
`summa.sh automake DIR`	genera la capa automake (`summa-automake.sh`)
`summa.sh rpm DIR`	genera el `.spec` (`summa-rpm.sh`)
`summa.sh crear DIR`	escribe un `.summa` nuevo desde un `debian/` completo
`summa.sh probar DIR`	ida y vuelta en un temporal, y compara

8. Armar los paquetes: sp paquete

```
sp paquete LUGAR/NOMBRE
```

Arma cualquier proyecto desde su `.summa` y su `MANIFIESTO`: ningún proyecto trae una receta propia (17 de noviembre de 2026; antes cada uno tenía su `hacer-pkg`). Los pasos son siempre los mismos:

1. Una copia limpia: exactamente lo que dice el `MANIFIESTO`, y antes la comprobación de que no quede nada afuera en silencio (sección 5).

2. La versión completa, `AÑO.MAYOR.MENOR.rREV-N`: la revisión de svn se calcula en la copia de trabajo y se escribe en la copia. Con cambios sin comitear lleva `+local` y avisa.

3. En la copia, el `.summa` genera la capa automake, `debian/` y el `.spec`; corre `FUENTES_PREPARAR` y queda hecho el `configure`. Esa copia es el `tar.gz` de las fuentes, reproducible: mismos archivos, mismo orden, dueño 0, fecha del changelog y `gzip -n`.

4. Lo que pide `BUILDDEPENDS` y no está instalado se dice antes de compilar, con la orden para instalarlo.

5. Compila, corre las pruebas e instala en un directorio temporal (`make install` con `DESTDIR`).

6. Arma el paquete en el formato de la máquina y lo deja todo al lado del proyecto, cada archivo con su `.md5`.

Máquina (según `/etc/os-release`)	Paquete
Debian, Ubuntu, Trisquel, Ututo 2027	`.deb` (`dpkg-buildpackage`), y de él un `.tgz` y un `.rpm` con los mismos archivos
Fedora	el `.rpm` propio, con marca `fedora` (`rpmbuild`)
openSUSE	el `.rpm` propio, con marca `suse`
Arch	`.pkg.tar.zst` de pacman
cualquier otra	`.tgz`, con su archivo de control

En todos los casos quedan además el `NOMBRE_VERSION.tar.gz` de las fuentes y el `NOMBRE_VERSION.summa` con la versión. Fuera de Debian y sus derivadas, el paquete se arma desde las fuentes y la instalación, no se convierte de un deb: cada paquete toma lo que dice su lista de instalación, y las dependencias se traducen con `diccionariodepaquetes` (entradas `FEDORA`, `SUSE`, `RPM`, `ARCH`). `SP_FORMATO=deb|rpm|pacman|tgz` fuerza el formato, para las pruebas.

Cuando algo falla, dice en qué paso, en qué línea y qué orden, y en los casos conocidos qué hacer. Los paquetes armados en la máquina propia sirven para probar y para instalar ahí; los que se suben son los de `sp binarios`. Construir no instala ni publica. Los detalles de cada formato están en el apéndice F.

El paquete mínimo. Si faltan las herramientas de construcción y no hay red para traerlas, `sp` arma a mano los paquetes de summapack (y de `mem_util` si hay compilador), sin dpkg-dev ni rpmbuild, e instala el que entienda el gestor de la máquina. Su versión lleva `~minimo`, para que el paquete completo lo reemplace en cuanto se pueda. El recorrido está en el apéndice G.

9. Binarios por distribución: sp binarios

```
sp binarios LUGAR/NOMBRE [ubuntu] [fedora] [suse] [ututo2027]
```

Arma los binarios de un proyecto en el Podman de cada distribución, con la cadena completa de nuestras dependencias construida ahí mismo. Usa las fuentes que dejó `sp paquete`, así que va después. Sin distribuciones, las cuatro. Cada distribución compila contra sus propias bibliotecas, y la marca va sólo en el nombre del archivo; la versión de adentro no cambia:

Distribución	Nombre
Ubuntu, Debian	`NOMBRE_V_ARQ.deb`
Ututo 2027	`NOMBRE_V+ututo2027_ARQ.deb`
Fedora	`NOMBRE-V.fedora.ARQ.rpm`
openSUSE	`NOMBRE-V.suse.ARQ.rpm`
Arch	`NOMBRE_V+arch_ARQ.tgz`

Un Podman limpio muestra lo que falta declarar en `BUILDDEPENDS`, y así se encontró casi todo lo que faltaba en noviembre de 2026. Las imágenes están en `distros/`: Ubuntu, Debian, Fedora, openSUSE y Arch, en versiones viejas (para que lo compilado ande también en las posteriores) y actuales; Trisquel 12, armada desde una raíz firmada; Ututo 2027, sobre Trisquel; y Linux From Scratch, para probar el tgz genérico. `sp/tests/test_generadores_podman.sh` prueba en cada imagen el `make install` con su desinstalador y el paquete de summapack en el formato de esa imagen.

Hasta octubre de 2026 se probó otro camino: un solo juego de paquetes compilado en una base vieja (Ubuntu 22.04), con las bibliotecas que cambian de número compiladas aparte en `/usr/lib/simusol` (el proyecto *comunes*). Queda en `distros/` para los casos que lo necesiten; el camino actual son las fuentes, y los binarios por distribución.

10. Publicar: sp subir y sp estable

El directorio de descargas está organizado por proyecto:

```
downloads/
  0-README-como-instalar  guia publica unica
  sp                 el sp autonomo: instala summapack y los proyectos
  calcula.summa      el .summa estable de cada proyecto, con su VERSION
  calcula/           los archivos estables (tar.gz, binarios)
    otros/           .md5, inestables, versiones anteriores
    fuentes-distro/VERSION/   .dsc, .debian.tar.xz, .orig.tar.gz, .src.rpm
```

```
sp subir --prueba LUGAR/NOMBRE        dice que subiria, sin subir
sp subir LUGAR/NOMBRE                 fuentes, .summa y binarios, como inestable
sp subir --tipo fuentes LUGAR/NOMBRE  solo el tar.gz y el .summa
sp subir --tipo distro LUGAR/NOMBRE   las fuentes por formato, tal cual
sp estable LUGAR/NOMBRE               lo inestable pasa a estable
sp subir --bootstrap LUGAR/summapack  el sp autonomo, la guia y los catalogos
```

`sp subir` exige el proyecto comiteado; si la versión actual no está armada, corre antes `sp paquete`. Lo inestable lo baja quien pide `-inestable`; `sp estable` pone arriba el `PROYECTO.summa` con la versión nueva, y antes comprueba que las dependencias nuestras estén estables en el servidor. Las fuentes por formato van aparte y sin pasar por inestable: el `.dsc` nombra a sus archivos por el nombre exacto, así que no se los puede renombrar, y cada versión queda en su carpeta. Si una distribución ya tenía binarios de un proyecto y faltan los de la versión nueva, `sp subir` los arma antes.

`SUBIR_SERVIDOR=local SUBIR_DIR=CARPETA` publica en una carpeta de esta máquina con la misma forma que el servidor: así se prueba un lanzamiento entero, y así arma `sp iso` las fuentes de la ISO.

10.1. Qué publica cada tipo y cómo probar la instalación por Internet

Opción de `sp subir`	Entrega en el servidor
`-tipo fuentes`	Tarball con las fuentes del proyecto y su receta `.summa` versionada.
`-tipo distro`	Fuentes GNU con `configure` en el `.orig.tar.gz`, archivos Debian `.dsc` y `.debian.tar.xz`, y `.src.rpm` cuando se construye con `rpmbuild`. Se publica por separado.
`-tipo deb`	DEB general y DEB identificado con la marca Ututo cuando están disponibles.
`-tipo rpm`	Binarios RPM. Fedora y SUSE reciben los construidos en sus respectivos entornos.
`-tipo tgz`	TGZ genérico y TGZ con marca Arch cuando están disponibles.
Sin `-tipo`	Selecciona fuentes, DEB, RPM y TGZ. Los nombres de los archivos que imprime la orden permiten comprobar cuáles encontró para esa versión.

En la prueba del 21 de noviembre de 2026, el constructor de Arch generó un `.pkg.tar.zst` válido como archivo de salida. El publicador rechazó `-tipo pacman` como tipo desconocido. El archivo probado con `-tipo tgz` en Arch llevaba el sufijo `+arch_all.tgz`.

Prueba reproducible con el catálogo del 21 de noviembre de 2026. El catálogo de pruebas está en `https://simusol.org/downloads/pruebas/sp-9387-ZfDVoRu4/`. Para usarlo con el cliente del árbol en una máquina Ubuntu, con SummaPack y sus generadores disponibles, ejecutar en Bash:

```
sitio=https://simusol.org/downloads/pruebas/sp-9387-ZfDVoRu4
```

```
sp=$HOME/svn/siru/summapack/sp/sp
BAJAR_URL="$sitio" "$sp" --sitio --para fuentes-paquete install SOURCE radia
dpkg-query -W radia
BAJAR_URL="$sitio" "$sp" --sitio --para deb install SUMMA exprtk
dpkg-query -W libexprtk-dev
```

La primera orden de instalación descarga fuentes, construye el DEB en la máquina destinataria e instala Radia. La segunda descarga el DEB publicado de ExprTk. `-sitio` selecciona Internet como origen y `BAJAR_URL` indica el catálogo para esa invocación. Por omisión se consulta `https://simusol.org/downloads/`.

- **Fuentes.** Se descargaron por HTTPS los 221 archivos de fuentes, recetas y sumas de los 17 proyectos de Siru y Calcula. Sus SHA-256 coincidieron con los de las copias locales.
- **Binarios.** Se publicaron los seis binarios de ExprTk correspondientes a las variantes DEB, RPM y TGZ de la tabla. Se descargaron los seis y se compararon sus SHA-256.
- **Instalaciones completas.** En contenedores Ubuntu descartables, `dpkg` confirmó Radia `2026.1.2.r9388-1` construido desde las fuentes publicadas y `libexprtk-dev` `2025.01.01.r9296-1` instalado desde el DEB publicado. El registro de estas pruebas acredita esas dos instalaciones. La instalación de los 17 proyectos en cada distribución sigue pendiente.

Orden de un lanzamiento completo.

1. `sp paquete` de cada proyecto, empezando por summapack y siguiendo por sus dependencias: unidades, exprtk, slex, tabular2, simplot, siru, lanzador, sceptresimusol, simusol, dia-simusol.
2. `sp binarios` de cada uno.
3. `sp subir -prueba` de cada uno; revisar.
4. `sp subir` de cada uno.
5. Probar la instalación en una máquina limpia (un Podman nuevo): `sp -para fuentes install simusol`.
6. `sp estable` de cada uno.
7. Si cambió summapack, `sp subir -bootstrap summapack`.

11. Instalar con sp

Para	Orden
instalar directamente desde las fuentes: `make install` en `/usr/local`, con lista para desinstalar	`sp -para fuentes install simusol`
lo mismo, en el home y sin root	`sp -para fuentes -prefijo ~/.local install simusol`
forma preferida cuando está disponible: compilar acá e instalar como paquete de la distribución	`sp -para fuentes-p install simusol`
el binario de esta distribución	`sp install simusol`
un binario de una distribución en particular	`sp -para ututo2027 install simusol` (o `deb`, `fedora`, `suse`, `arch`)
ver qué haría, sin hacerlo	`sp -ensayo ...`
desinstalar	`sp remove simusol`
lo instalado desde fuentes	`sp list-sour`
buscar, informar	`sp search TEXTO`, `sp info PAQUETE`
desde el árbol de svn, sin el servidor	`sp -para fuentes install simusol` (ya es lo que

En una máquina sin summapack, el `sp` autónomo hace lo mismo sin la palabra `install`: `sh sp -para fuentes simusol`. Lleva summapack adentro, así que instala aunque no haya red ni servidor.

`-para fuentes-paquete` es lo que hace un usuario de Ututo 2027: baja la fuente completa de cada proyecto, hace `make install` en un directorio, arma con eso el paquete de la distribución y lo instala con su gestor, en orden y con toda la cadena. `-para fuentes` instala sin paquete y deja registro: cada instalación desde fuentes (y cada `.tgz`) deja un desinstalador en `PREFIJO/var/lib/summapack/desinstalar/`, y lo mismo un `make install` directo de un proyecto con la capa automake.

`sp` baja todo con `curl`, y lo dice si falta. Se baja cada archivo con su `.md5`, por https; si no coincide, se descarta. El servidor se elige con `BAJAR_URL`, que acepta `file://` para probar.

Desde el árbol, sin internet. `svn/siru/summapack/sp/sp`, sin ninguna orden, instala o actualiza summapack desde el árbol: revisa e instala sólo las herramientas que faltan para fabricar el paquete, lo arma con `sp paquete` y lo instala. Si no hay herramientas ni red, queda el paquete mínimo (sección 8).

12. Ututo 2027

Ututo 2027 es una distribución basada en Trisquel 12 Ecne, para simulación, cálculo y escritura científica. Usa el formato deb por debajo, pero **su gestor de paquetes es `sp`**, que en muchos casos es una interfaz de apt: los programas de la distribución los instala apt, y los nuestros `sp`, desde las fuentes. Lo cuenta `siru/ututo/GESTOR-DE-PAQUETES.md`.

```
sp iso ututo2027
```

Arma la ISO desde nuestras fuentes completas, como las instalaría el usuario (17 de noviembre de 2026):

1. publica, con la forma del servidor de descargas, las fuentes de cada proyecto de `siru/ututo/proyectos-simusol.list` (las de `sp paquete`) y el `sp` autónomo;
2. arma la raíz de Trisquel e instala los programas de la distribución (curl entre ellos, porque `sp` baja todo con él);
3. adentro, instala cada proyecto con `sp -para fuentes-paquete install`;
4. deja esas fuentes en la ISO, en `/usr/src/ututo2027` del sistema y en `fuentes/ututo2027` del disco: son las fuentes exactas de lo instalado, y con ellas `sp` puede recompilar sin internet.

La ISO no se corta por el estado del árbol (18 de noviembre de 2026): tarda horas y el árbol sigue cambiando, así que nunca lleva la última versión de todo. Si un proyecto tiene cambios sin comitear o archivos sin declarar en su `MANIFIESTO`, `sp iso` avisa y arma con lo que hay (la versión queda marcada `+local`); y si un proyecto igual no se puede armar, van sus fuentes de la publicación anterior, y lo dice. `sp subir`, para el servidor, sigue exigiendo todo comiteado.

`sp iso ututo2027 -preparar`, `-reanudar`, `-solo-iso` o `-disco-vm` hacen sólo esa etapa. Lo demás de la distribución (el arte, el instalador, el pendrive) está en `siru/ututo/README.md`. Los binarios de Ututo 2027 llevan su marca (`+ututo2027`) y se arman con `sp binarios NOMBRE ututo2027`.

13. La capacidad Ajolote: un requisito de los paquetes modernos

Agregado el 5 de noviembre de 2026, a pedido del autor, a partir del caso de `sceptresimusol`.

Todo paquete binario muere al cambiar el sistema operativo: queda atado al compilador y a las bibliotecas con que se construyó. Lo habitual es que el sistema nuevo traiga otro paquete construido para él, pero eso supone que alguien lo va a construir y distribuir, y nada lo asegura. Para un programa mantenido por un grupo chico, suponerlo es perder el programa.

De ahí el requisito: **cada paquete binario debe traerse sus propias fuentes y poder rehacerse a sí mismo**. En términos concretos, el binario instala en `share/PAQUETE/fuentes` el `tar` de sus fuentes y su `.summa` —lo que `sp` necesita para rearmarlo—, y el programa avisa, cuando detecta que el entorno cambió, con la orden exacta:

```
sp --para /usr/share/PAQUETE/fuentes install PAQUETE
```

A esa propiedad el autor la llama **capacidad Ajolote**: el ajolote se reconstruye desde casi cualquier parte, siendo un animal complejo. La *Hydra* común lo hace más —y además no muere en condiciones favorables—, pero un programa como SCEPTRE es complejo, así que se parece más a un ajolote.

13.1. El paralelo con DKMS

La forma es la misma que usa DKMS (*Dynamic Kernel Module Support*) para los controladores que no están en el núcleo oficial: el paquete no instala el módulo compilado, instala sus fuentes en `/usr/src/NOMBRE-VERSION/` con un `dkms.conf`, y el sistema lo recompila al instalar un núcleo nuevo. Tres diferencias:

- DKMS sirve sólo para módulos del núcleo; esto es para programas de espacio de usuario.
- DKMS lo dispara el sistema; acá avisa el propio programa y la reconstrucción la ordena la persona.
- DKMS existe en las distribuciones de Linux (Debian, Fedora por EPEL, SUSE, Arch) y no fuera de ellas; esta reconstrucción la hace `sp`, que arma `.deb`, `.rpm` y `.tgz`.

Es la continuación natural del *Manifiesto Anti-distros* [1]: si el empaquetado es parte del desarrollo y no de la distribución, entonces el paquete no puede depender de que una distribución lo reconstruya. El texto para incorporar al manifiesto, como requisito necesario de los paquetes modernos, está en el apéndice E: el `doc/UTUTO-ANTIDISTROS.html` de este directorio es una captura de la página publicada y no se edita acá.

13.2. Dónde está puesto (5 de noviembre de 2026)

Lo tienen los paquetes de estos árboles que traen binarios compilados, que son los que mueren al cambiar el sistema:

Paquete	Cómo
`sceptresimusol`	Completo: las fuentes, la huella (`gfortran-mod`) y el aviso en `sc1`, porque compila Fortran en cada corrida.
`slex`	Las fuentes en `share/slex/fuentes`.
`tabular2`	`share/tabular2/fuentes`.
`unidades`	`share/unidades/fuentes`.
`calcula`	`share/calcula/fuentes`.
`libmem-util`	`share/mem_util/fuentes`, con el `.summa` de summapack, que es quien la empaqueta.

En todos, junto al `tar` y al `.summa` queda un `COMO-REHACER` con la orden exacta. Los únicos con huella y aviso son los que compilan algo en la corrida: en los demás no hay un momento en que se note solo, y de que la `libc` o el *soname* cambiaron avisa `dpkg` al no poder instalar.

Los paquetes sin binarios —`siru` y `simplot` (Perl), `dia-simusol` (Python y Dia), `puyuspa` (web), el propio `summapack`— no lo necesitan: son independientes de la arquitectura y no los rompe un compilador nuevo.

13.3. Qué hace falta en el paquete

Las fuentes adentro: el `tar` que produce `make dist` y el `.summa`, en `share/PAQUETE/fuentes`. Lo que no se necesita para construir —manuales escaneados, por ejemplo— se saca con un `dist-hook`: en `sceptresimusol` eso bajó el `tar` de 50 MB a 1,3 MB.

La huella del entorno: qué hay que comparar para saber que el sistema cambió. En `sceptresimusol` es `share/PAQUETE/gfortran-mod`, la versión de los `.mod` de gfortran con que se construyó, porque el programa compila Fortran en cada corrida.

El aviso: el programa compara la huella y, si no coincide, explica la causa y da la orden. No debe costar nada en el uso normal.

Que falle a tiempo: además, el paquete declara la dependencia que lo vuelve no instalable con un entorno incompatible (en Debian, el paquete virtual `gfortran-mod-N`), así el aviso llega al instalar y no en medio de un trabajo.

14. Dónde van los manuales, los lanzadores y los iconos

Agregado el 5 de noviembre de 2026, decisión del autor.

14.1. Los manuales: `usr/share/doc/simusol/PAQUETE`

La documentación de **todos** los programas de los árboles `siru` y `calcula` va al mismo lugar, `usr/share/doc/simusol`, en un subdirectorio por programa, y no al `doc/` de cada paquete. Así la documentación de Simusol queda junta en vez de repartida en un directorio por paquete, y de ahí la lista el menú Documentos de los programas que lo tienen.

No es nuevo: lo hacía `dia-simusol` desde antes, y su `.summa` lo dice con todas las letras. Lo que se hizo fue extenderlo a los demás:

Programa	Qué pone en `usr/share/doc/simusol/...`
`dia-simusol`	`doc/help` y `manual.pdf`
`siru`	`doc/*.md` y los ejemplos
`simplot`	el `doc/` entero
`tabular`	`RESULTADOS.md`
`calcula`	`ESCRIBIR-MODELOS.md`, `MODULODINAMICO.md`, `PROTOCOLO-MAQUINA.md`
`puyuspa`	`DOCUMENTACION.md`, `PLAN.md`
`sceptresimusol`	el manual en PDF

Dos cuidados que ya costaron un error cada uno:

Lo que la política de Debian pide en el `doc/` del paquete —`copyright`, `changelog`, y de paso `README` y `NEWS`— se queda ahí. Sólo se mudan los manuales.

`dh_compress` comprime los `.pdf` y los deja como `manual.pdf.gz`, con lo que ni el visor ni un lanzador que nombre el archivo lo abren. El paquete que instale un PDF necesita `override_dh_compress: dh_compress -X.pdf`.

En los paquetes que empaqueta un `.summa` con `docs`, los manuales pasan de esa lista —que siempre usa el `doc/` del paquete— a la lista de instalación, donde se puede decir el destino.

14.2. La documentación se instala hecha

Decisión del autor del 16 de noviembre de 2026. Lo que el `.summa` manda a `usr/share/doc/simusol` —lo que lista el menú Documentos— se convierte al construir, y se instala sólo lo generado:

Fuente	Se instala	Con
cada `.md`	su `.html`, con los enlaces a otros `.md` pasados a `.html`	`pandoc`; sin él, el texto dentro de un HTML, con un aviso
cada `.tex` con `\documentclass`	su `.pdf`, y su HTML: una carpeta `NOMBRE-html` con una página por sección	el PDF, con el `Makefile` de su carpeta si el `MANIFIESTO` lo trae, o `pdflatex` dos veces; el HTML, con `make4ht` (tex4ht); sin él, una página que lleva al PDF, con un aviso
los `.tex` sin `\documentclass`	nada: son partes del principal y van dentro del PDF	

Todo documento de los proyectos está en LaTeX y se ve como PDF y como HTML (17 de noviembre de 2026). El HTML usa dos archivos comunes a todos los proyectos: `simusol-doc.cfg`, la configuración de tex4ht (la navegación entre páginas, en castellano), y `simusol-doc.css`, la hoja de estilo. La capa automake los copia a las fuentes de cada proyecto que tenga documentos en LaTeX, así que viajan con ellas, y compila el HTML en una copia aparte, con babel en castellano y la opción `es-nolayout`, sin la cual tex4ht arma mal los identificadores de los apéndices.

Lo hace la capa automake (`instalar.am`), así que ningún proyecto tiene que escribirlo, y las listas del deb, del rpm y de los demás formatos piden lo generado. Por eso cada proyecto con documentación pide `pandoc`, y texlive si tiene LaTeX (con `texlive-extra-utils` y `texlive-plain-generic`, de donde salen make4ht y tex4ht), en sus `BUILDDEPENDS`. Este manual es un ejemplo: se instala como `usr/share/doc/simusol/summapack/summapack.pdf` y como `summapack-html/summapack.html`, con el paquete `summapack-doc`, que se arma aparte (el proyecto `summapack/doc`) para que summapack, la capa básica, no necesite LaTeX.

14.3. Los lanzadores y los iconos

Un programa con interfaz instala sus entradas `.desktop` en `usr/share/applications` y su icono en `usr/share/icons/hicolor/scalable/apps`: de ahí las toma el menú de aplicaciones del escritorio, sin que haga falta ningún `postinst`. El SCEPTRE viejo lo hacía al instalar, con un `gnome_icon_install` que ya no existe en ninguna parte.

Ponerlas en el escritorio *de un usuario* es otra cosa, y la hace el `user setup` de cada programa (en SCEPTRE, `sceptre-user-setup`), que corre el usuario cuando quiere y no la instalación: busca el escritorio con `xdg-user-dir` —con `~/Escritorio` y `~/Desktop` como respaldo—, copia los `.desktop` y, en GNOME, los marca de confianza con `gio set ... metadata::trusted`.

Una entrada cuyo `Exec` nombre una ruta que depende del prefijo (el manual, por ejemplo) se genera al construir desde un `.desktop.in`, y la regla tiene que depender del `Makefile`, porque de ahí sale la ruta. Y conviene pasar el resultado por `desktop-file-validate`: encontró, por ejemplo, que `%h` no es un código válido de `Exec`.

15. Publicar documentos: subir-doc

`subir-doc` publica los dos PDF de SummaPack. No modifica páginas de CMSimple, paquetes ni versiones estables o inestables. Se instala en `/usr/bin/subir-doc`; en las fuentes está en la raíz del proyecto. Los PDF de origen no se incluyen en el paquete.

15.1. Uso y archivos

```
cd ~/svn/siru/summapack
(cd doc && latexmk -pdf summapack.tex)
./subir-doc --prueba
./subir-doc

# Comando instalado, desde cualquier directorio:
subir-doc --doc ~/svn/siru/summapack/doc --prueba
subir-doc --doc ~/svn/siru/summapack/doc
```

Opción	Efecto
`-doc DIRECTORIO`	Busca allí `summapack.pdf` y `738.pdf`; no cambia el proyecto destino.
`-prueba`	Valida los PDF y muestra las rutas; no conecta, no escribe y no verifica permisos ni existencia del proyecto remoto.
`-h`, `-help`	Muestra la ayuda sin publicar.

Sin `-doc`, usa `doc/` junto al script, si existe; en caso contrario, `doc/` del directorio de trabajo. Los dos PDF son obligatorios y se comprueba su cabecera. No acepta archivos sueltos ni publica automáticamente otros PDF. Si existe `summapack.tex` y es más nuevo que su PDF, pide regenerar el manual.

Origen	Nombre público en `downloads/summapack/docs/`
`summapack.pdf`	`summapack-2026.pdf`
`738.pdf`	`sumapack-jsl-2011.pdf`

15.2. Configuración y requisitos

Requiere Bash, rsync y SSH. Lee `~/.config/simusol/subir`, como `subir-paquetes`; las variables fijadas en el entorno tienen prioridad. La configuración es un archivo de shell de confianza del usuario:

```
SUBIR_SERVIDOR=simusol.org
SUBIR_USUARIO=publicador
SUBIR_DIR=/var/www/simusol.org/public_html/downloads
```

`SUBIR_DIR` es la raíz de descargas: el comando agrega `summapack/docs/`. Debe ser una ruta absoluta, sin espacios ni componentes `.` o `...` El usuario remoto necesita escritura en el proyecto y acceso SSH sin interacción; el comando no solicita sudo ni contraseñas.

El directorio del proyecto debe existir. Si existe `downloads/summapack/`, crea `docs/` cuando falta. Si falta el proyecto, se detiene sin crearlo ni subir archivos. Tampoco acepta `docs` como enlace simbólico ni como archivo ordinario. La condición también vale en modo local.

Prueba real, pero solo en un directorio temporal local:

```
prueba_docs=$(mktemp -d /tmp/summapack-docs-prueba.XXXXXXXX)
mkdir -p "$prueba_docs/summapack"
SUBIR_SERVIDOR=local SUBIR_DIR="$prueba_docs" ./subir-doc
```

15.3. Actualizaciones, respaldos y fallos

- No vuelve a transferir archivos cuyo contenido no cambió. Conserva los PDF reemplazados en `docs/.anteriores/FECHA-ID/`.
- No borra documentos ajenos ni paquetes. Los respaldos no se eliminan automáticamente; para recuperar uno, revisar y copiar el PDF elegido al nombre público.
- Usa archivos temporales de rsync antes de reemplazar cada documento. No es una transacción conjunta de ambos PDF.
- Devuelve 0 si termina correctamente; 2 para opciones o configuración inválidas; 1 para comprobaciones fallidas de origen o destino. Los errores de SSH o rsync conservan su estado no nulo. Corregir la causa y repetir; ante un fallo, algún PDF puede haber sido actualizado.

En TinyMCE, seleccionar el texto y usar *Insertar enlace* con una de estas direcciones:

- `/downloads/summapack/docs/summapack-2026.pdf`
- `/downloads/summapack/docs/sumapack-jsl-2011.pdf`

Los enlaces relativos funcionan en los tres dominios. Subir el PDF no crea una página ni su entrada de menú.

Las pruebas locales se ejecutan con `python3 pruebas/prueba-subir-doc.py`, sin conectarse al servidor.

16. Páginas web: herramientas y lenguajes

Las herramientas propias pertenecen a SummaPack. La entrada única es `sincronizar-pagina SISTEMA [-prueba]`; todos los demás datos están en `share/summapack/data/paginas-web.json`, sustituible por `~/.config/simusol/paginas-web.json`. La tabla agrupa las páginas de cada sistema y define conexión, MD, HTML de salida, base y política inicial.

Programa	Función	Estado e implementación
`sincronizar-pagina`	Consultar la web y coordinar MD, HTML, subida y bajada por sistema.	Implementado: lanzador Bash, tabla JSON, motor Python 3 y SSH. Solo sistema y opción `-prueba`.
`subir-pagina`	Publicación directa y opciones avanzadas.	Conservado por compatibilidad; no se necesita en el trabajo habitual.
Bajada de páginas	Página CMSimple a HTML local y MD.	Operación interna del comando único; no requiere otro programa.
`pandoc`	MD a HTML y HTML descargado a MD.	Dependencia externa escrita en Haskell, invocada como ejecutable; no es código propio de SummaPack.

La web se consulta antes de regenerar HTML. Si cambió solo MD, genera y sube; si cambió solo la web, descarga e incorpora al MD; si cambiaron ambos, se detiene sin sobrescribir. El HTML local se genera o se descarga, no es una tercera fuente editable. Bash lee la configuración; Python compara versiones, usa SSH y administra respaldos y archivos. No se necesita programar en Haskell para mantener estas herramientas. La guía completa de configuración, respaldo y límites está en el apéndice K.

16.1. Orden y referencia de sincronización

1. Consultar primero la página web; si no se puede leer, detenerse sin regenerar HTML.
2. Comparar el MD y la web con sus respectivas versiones en la última base común, no por fechas.
3. Decidir: nada cambió, no publicar; solo MD, generar y subir; solo web, descargar e incorporar al MD; ambos o sin base, detenerse para revisión explícita.
4. Preparar las conversiones necesarias y respaldar lo que se reemplazará.
5. Aplicar la dirección elegida. Antes de publicar, verificar bajo bloqueo que la huella web siga siendo la consultada; antes de importar, comprobar que el MD no haya cambiado entretanto.
6. Guardar la nueva base local solo después de completar la actualización y sus archivos locales. Ante un fallo parcial, conservar la base anterior y revisar los respaldos; no existe una transacción distribuida que deshaga todo.

La base de `-base` guarda un par correspondiente de versiones MD y HTML web, junto con la identidad de la página y sus rutas. No compara Markdown directamente con HTML. El estado remoto conserva la huella de publicación; el HTML local es solo una salida generada o descargada. Si no cambió ninguna fuente, puede recuperarse ese HTML si falta, sin publicar ni modificar el MD.

`-prueba` consulta y prepara diferencias sin reemplazar archivos ni crear base. No hay ejecución periódica: se revisan las páginas configuradas del sistema y, sin conflictos, se aplican una por una volviendo a consultar. No es una transacción conjunta entre páginas. Los generadores antiguos siguen pendientes de migrar: un `generar.py` aparte no consulta automáticamente la web.

```
sincronizar-pagina slex --prueba
sincronizar-pagina slex
```

La tabla inicial incluye Slex y la portada de SummaPack con inicio desde web, Tabular2 con revisión inicial y las entradas de Calcula y Puyuspa pendientes de definir fuente y destino. Si se importa desde web, se puede crear el MD que falta; no se crean páginas del CMS. Las subpáginas no configuradas no se modifican. Consultar el apéndice K para el esquema y los límites.

17. Por dentro: referencia de los componentes

Para quien mantiene summapack. Quien lo usa necesita sólo `sp`: lo que sigue es lo que `sp` llama por debajo, y no se documenta para el usuario ni se le pide que lo llame. Esta referencia describe las herramientas del árbol actual, no una promesa de que todos los sistemas o proyectos estén soportados. Los nombres instalados de los scripts de `bin/` pierden las extensiones `.sh` y `.pl`; `distro.sh` se instala además como alias histórico. La biblioteca y los catálogos están en `/usr/share/summapack/data/` con prefijo `/usr`. Para una instalación desde fuentes, sustituir ese prefijo por el elegido.

17.1. Elegir una operación

Necesidad	Componente y efecto
Consultar	`summapack`, `distro`: identifican versión, rutas y sistema; no instalan.
Obtener e instalar	`sp`; por debajo, `bajar-paquetes` e `instalar-tgz`: descargan y, cuando corresponde, modifican la máquina.
Preparar dependencias	`preparar`, `packinstall`, `fuentesexternas`, `diccionariodepaquetes`: dos rutas distintas, fuentes o paquetes del sistema.
Construir	`sp paquete`, `sp binarios`, `sp iso`; por debajo, `summa.sh` y sus generadores, `armar-proyecto`, `hacer-pkg.inc`, `makefile.lib`, `distros/`.
Publicar	`sp subir`, `sp estable`; por debajo, `subir-paquetes`. `subir-doc`: escriben en el destino configurado; no editan CMSimple.
Integrar el escritorio	`icon_install`, `findbinlib`: lanzadores y diagnóstico de instalaciones superpuestas.
Biblioteca C	`mem_util`: memoria, archivos, rutas, cadenas y fechas; paquetes de ejecución y desarrollo.
Documentar y probar	`doc/web/`, `pruebas/`: generan y comprueban documentación y casos controlados; no publican automáticamente.

17.2. summapack y distro

```
summapack
summapack --version
summapack --help
distro
distro +
distro --help
```

`summapack` informa la versión registrada por dpkg o rpm y busca sus datos en `SUMMAPACK_DATA`, `/usr/share/summapack/data`, `/usr/local/share/summapack/data` y el directorio personal. Una instalación solo desde fuentes o TGZ puede aparecer con versión desconocida: no significa que sus archivos estén ausentes.

`distro`, versión 4 del script, consulta `uname -s`, no el nombre del equipo. En Linux lee `/etc/os-release`: primero `ID` y, si no lo reconoce, `ID_LIKE` en el orden declarado. Solo usa `/usr/lib/os-release` si falta el primero; no los mezcla. Lee datos sin ejecutar su contenido. Sin identificación moderna consulta `lsb-release` y archivos históricos. Con `+` agrega nombre y número de versión: `UBUNTU_noble_24_04`.

openSUSE y SLES se agrupan como `SUSE`, Manjaro como `ARCH`, Rocky y AlmaLinux como `REDHAT`. WSL devuelve su distribución real. BSD, macOS y Solaris tienen identificadores propios. Linux no reconocido devuelve `OTHERGNULINUX`; otro kernel desconocido, `UNKNOWN`. Un argumento inválido termina con estado 2. `DISTRO_ROOT=/ruta` permite consultar archivos de otra raíz manteniendo el kernel actual, para diagnóstico: no es un chroot. La familia detectada no garantiza compatibilidad binaria ni decide el formato de `bajar-paquetes`.

17.3. sp y el motor interno de descarga

Hay cuatro entradas con responsabilidades diferentes:

Orden	Intención y momento de uso
`sh sp -para deb calcula`	Arranque público: instala SummaPack, Calcula y sus dependencias en una sola orden.
`sp install calcula`	Interfaz habitual una vez instalado SummaPack. Reconoce proyectos Simusol y usa `bajar-paquetes`; para dependencias externas utiliza el gestor de la distribución.
`packinstall calcula`	Compatibilidad con scripts anteriores. Adapta la sintaxis antigua a `sp`; no constituye otro motor de instalación. Para código nuevo, preferir `sp`.
`bajar-paquetes`	Motor reutilizable de descarga. Permite elegir formato, versión, fuentes o un paquete individual; sin `-instalar` descarga y verifica, pero no instala.

El archivo público `sp` incorpora su motor de arranque; una vez instalado, la misma orden `sp` administra los paquetes. `packinstall` conserva llamadas internas antiguas. `bajar-paquetes` es un componente interno, no otra instrucción de instalación para usuarios. Sin `-para`, el arranque ofrece elegir el formato. `-prueba` permite revisar el recorrido sin instalar, aunque puede consultar metadatos o crear temporales.

```
# Descargar solamente un paquete binario:
bajar-paquetes --para deb tabular2/tabular
# Instalar un proyecto con la interfaz publica:
sp install tabular2
# Compilar e instalar desde fuentes en un prefijo propio:
sh sp --para fuentes --prefijo "$HOME/.local" tabular2
```

Opción de bajar-paquetes	Uso
`-para`, `-formato`	`fuentes`, `deb`, `fedora`, `suse`, `arch`, `rpm`, `tgz`, `fuentes`, `tar.gz` o `summa`. Sin opción y sin terminal, falla.
`-instalar`	Instala binarios con el gestor correspondiente, o compila e instala si se eligió `fuentes`.
`-inestable`, `-version V`	Seleccionan la entrega. Sin ellas se pide la estable. Si no hay inestable, se vuelve a la estable.
`-dir D`	Directorio de descarga, normalmente `~/.cache/summapack`. También `BAJAR_DIR`.
`-prefijo P`	Destino de instalación desde fuentes; por omisión `/usr/local`. No cambia el destino interno de un DEB o RPM.
`-solo-dependencias`	Con `-para fuentes -instalar`, prepara lo necesario sin construir el proyecto solicitado.
`-sin-dependencias`	En el flujo binario, descarga solo lo pedido; no es garantía de que el paquete pueda instalarse o ejecutarse.
`-prueba`	Muestra operaciones, pero no es un entorno aislado ni garantiza ausencia de descargas de metadatos o de archivos temporales. Su alcance depende del flujo.

La entrada puede ser `PROYECTO`, `PROYECTO/PAQUETE` o un `ARCHIVO.summa`. El servidor se configura con `BAJAR_URL`, también en `~/.config/simusol/subir`. Los binarios y fuentes propios se comprueban con MD5; las fuentes externas, con SHA256. Estas sumas no son una firma del autor: debe confiarse en el servidor y los metadatos. Los paquetes específicos de distribución se intentan antes que los generales; un reemplazo general compilado en Ubuntu no garantiza compatibilidad con Fedora o Arch.

El flujo desde fuentes usa `BUILDDEPENDS`, `PKGCONFIG`, `CONFIGURE`, `FUENTES_CONSTRUIR` y `FUENTES_INSTALAR`. Compila primero dependencias propias que no detecta mediante pkg-config y busca las externas en el catálogo. La presencia de un módulo pkg-config no comprueba todas las restricciones de versión. No es un resolvedor universal de dependencias ni un registro equivalente a dpkg: la instalación directa desde fuentes no tiene el mecanismo `-quitar` de TGZ. Los campos de construcción son órdenes ejecutables de confianza; pueden requerir sudo al instalar. Los errores indican el temporal y su `compilar.log`. El flujo actual toma la versión declarada por el `.summa`; no suponer que las opciones de selección de binarios resuelvan del mismo modo cualquier construcción desde fuentes.

17.4. instalar-tgz

```
instalar-tgz --prueba paquete_VERSION_ARQ.tgz
instalar-tgz --raiz /ruta/imagen paquete_VERSION_ARQ.tgz
instalar-tgz --lista
instalar-tgz --archivos paquete
instalar-tgz --prueba --quitar paquete
```

Instala los TGZ de este proyecto y conserva en `RAIZ/var/lib/summapack/tgz/` la lista, versión y respaldo. Actualizar retira archivos que ya no trae la nueva versión; quitar elimina los registrados y restaura los respaldados. En la raíz del sistema consulta dpkg/rpm para evitar pisar archivos de otros paquetes, salvo `-forzar`; en una raíz alternativa no consulta su base de paquetes. Necesita Bash, tar/gzip y utilidades de archivos; para modificar rutas del sistema necesita permisos. No instalar archivos desconocidos: no es un sandbox, ni se promete una transacción con recuperación automática ante cualquier fallo. `-prueba` puede crear temporales aunque no haga la instalación.

17.5. preparar, fuentesexternas y dependencias

```
# Desde las fuentes del proyecto:
./preparar --prefijo "$HOME/.local" --prueba
./preparar --prefijo "$HOME/.local"
# En un proyecto que incluye makefile.lib:
make dependencias PRUEBA=1
make dependencias-distro PRUEBA=1
```

`preparar` vive en `share/summapack/data/preparar`; `hacer-pkg.inc` lo incorpora a los archivos de fuentes. No es necesariamente una orden instalada en el PATH. Busca el `.summa` primero en el directorio actual y después junto al script, descarga `bajar-paquetes` y, si no puede, intenta el instalado. Le pasa `-para fuentes -instalar -solo-dependencias`. No compila el proyecto principal; después se siguen sus instrucciones específicas. El ejemplo autotools no sirve sin adaptación para todos los proyectos.

`fuentesexternas` es un catálogo de recetas, no un ejecutable. Cada sección `[nombre-de-dependencia]` puede declarar `proyecto`, `version`, `url`, `sha256`, `comprobar`, `depende`, `configure`, `construir`, `instalar`, `base` y `origen`. Las entradas `base: si` deben estar previamente instaladas: el programa no construye todo el sistema desde cero. Las órdenes del catálogo se ejecutan; usar solo recetas confiables. Incluye autotools, pkgconf, Bison, bibliotecas de texto y recetas para CMake, GSL, OpenBLAS y SuiteSparse; que exista una receta no equivale a que toda combinación esté probada. Si falta una dependencia de base o una receta, se informa; puede ser necesaria una instalación manual.

17.6. packinstall y diccionariodepaquetes

```
packinstall --prueba UBUNTU libreadline-dev
packinstall --prueba libreadline-dev
packinstall --prueba calcula
packinstall --man
```

`packinstall` es ahora un adaptador Bash de la sintaxis histórica: acepta `[DISTRO] PAQUETE...`, módulos `Modulo::Perl` y entradas `REPO-ppa:usuario/repo`, y los remite a `sp`. Cuando el nombre corresponde a un proyecto Simusol, `sp` invoca `bajar-paquetes`; para dependencias externas invoca el gestor de la distribución. Los módulos Perl se buscan primero como paquete de la distribución y, si no hay equivalencia, mediante cpanm. Algunas distribuciones admitidas antiguamente no tienen packlet y producen un error explícito. `-prueba` se traduce al ensayo de `sp`. El programa Perl anterior permanece solo en `viejo/bin/packinstall.pl`.

`diccionariodepaquetes` contiene líneas `NOMBRE_DEBIAN:DISTRO:NOMBRE_ALLA`; `-` señala que no hace falta instalarlo en ese destino. Contiene además equivalencias para Perl y RPM. Lo consume `sp`, al que delega `packinstall`; también lo usan el empaquetado RPM y el instalador de dependencias de TGZ. No es el catálogo de fuentes: traduce nombres, no versiones ni compatibilidad ABI.

17.7. summa.sh y la receta de sp paquete

```
bash share/summapack/data/summa.sh campo . VERSION
bash share/summapack/data/summa.sh dependencias .
bash share/summapack/data/summa.sh probar .
```

Orden de summa.sh	Entrada, resultado y precaución
`crear DIR`	Importa `debian/` a `NOMBRE.summa`; rechaza un archivo ya existente.
`actualizar DIR`	Migra un `.summa` antiguo desde `debian/`, conserva campos adicionales y puede renombrar el archivo. Revisar el diff.
`debian DIR`	Genera metadatos y archivos de `debian/`; conserva el changelog. Sobrescribe archivos generados, por lo que se edita el `.summa`.
`probar DIR`	Hace una ida y vuelta en un temporal y compara; estado no nulo si hay diferencias. No construye binarios.
`campo DIR CAMPO`	Imprime un campo del `.summa`.
`dependencias DIR`	Lista nombres de `BUILDDEPENDS`, sin restricciones de versión y tomando la primera alternativa; no instala.

`summa.sh` es Bash y lee el `.summa` como shell: incluso una consulta debe hacerse solo sobre archivos confiables. El `.summa` contiene versión, paquetes, dependencias, recetas, archivos de instalación y metadatos; no es JSON ni un documento inerte.

La receta de `sp paquete` es `share/summapack/data/armar-proyecto`, una sola para todos los proyectos (17/11/2026: antes cada uno tenía su `hacer-pkg`); usa las funciones de `hacer-pkg.inc` y, fuera de Debian y sus derivadas, `armar-nativo.inc`. Los paquetes se copian al lado del proyecto y se anotan en `subir.lista`. Construir no instala ni publica. `hacer-deb.inc` queda como enlace a `hacer-pkg.inc`, para los proyectos viejos que todavía traen su receta (puyuspa-6.0).

17.8. Publicadores y configuración

`subir-paquetes [-prueba] [-forzar] ARCHIVO...` publica los paquetes y sus metadatos como inestables; `-estable PROYECTO` promueve la entrega y mueve las anteriores; `-lista` consulta el destino. `-reorganizar` cambia la organización del directorio de descargas: no es una

operación rutinaria. La asociación paquete/proyecto se obtiene del `.summa`, que debe acompañar los archivos o estar accesible. Requiere Bash, SSH/scp y utilidades de sumas y archivos. Con `-prueba` no realiza las escrituras previstas, pero algunas operaciones consultan el destino por SSH.

`subir-doc` es independiente: publica solo los dos PDF de SummaPack en `summapack/docs/`, crea únicamente ese subdirectorio si el proyecto ya existe y respalda los reemplazos. Sus opciones, validaciones y recuperación se explican en la sección dedicada. A diferencia del publicador de paquetes, su `-prueba` no conecta.

Ambos leen `SUBIR_SERVIDOR`, `SUBIR_USUARIO` y `SUBIR_DIR`; para este servidor se configura el usuario `publicador`. No asumir que el valor por defecto de un script tiene permisos SSH. El archivo `~/.config/simusol/subir` contiene código shell de confianza y puede afectar también al descargador. `sp subir -bootstrap summapack` publica `0-README-como-instalar`, el `sp` autónomo, `diccionariodepaquetes` y `fuentesexternas`; requiere autorización de publicación y no tiene un modo `PRUEBA=1` equivalente al de otras órdenes.

17.9. icon_install, findbinlib y templatesumma

```
icon_install --prueba /ruta/lanzadores
findbinlib "interpola" "" "" "" tabular
templatesumma --help
```

`icon_install [-prueba] [USUARIO] DIRECTORIO` procesa lanzadores del directorio, los agrega al menú mediante XDG y los copia al escritorio del usuario. Para otro usuario requiere root; para uno mismo, permisos locales. Necesita Bash, getent y herramientas XDG. Los errores de algunas operaciones de escritorio se toleran: terminar con 0 no demuestra que todos los lanzadores se hayan instalado.

`findbinlib "PROGRAMAS" "BIBLIOTECAS" "BORRA" "MODULOS_PERL" PAQUETE` busca instalaciones manuales que pueden ocultar las de un paquete. Son cinco argumentos; para solo inspeccionar se pasa una cadena vacía en el tercero. Devuelve rutas separadas por punto y coma. Si el tercer argumento es literalmente `BORRA`, mueve archivos y páginas de manual a `/tmp/PAQUETEmoved`; no es una limpieza para ejecutar a ciegas. Incluso la consulta crea allí un directorio auxiliar. Usa Bash, Perl y herramientas de archivos; tiene limitaciones históricas con espacios en nombres.

`templatesumma PAQUETE SVVERSION PACKAGER LICENCIA LIBMAYVER VERSION` es una herramienta Perl heredada con Template Toolkit. Desde el directorio de trabajo toma `template.changes`, `template.spec` y `debian/changelog.template`, y escribe `PAQUETE.changes`, `PAQUETE.spec` y `debian/changelog`. Puede sobrescribirlos; no tiene `-prueba`. Admite `-help`, `-man` y `-version`. Ejecuta plantillas de confianza, incluso código Perl; no es el flujo recomendado para proyectos nuevos, que usan `summa.sh`.

17.10. distros, mem_util, configuración y pruebas

`sp binarios` usa `distros/armar-paquetes`, y `sp iso` usa `distros/armar-iso`. `distros/compilar fedora|suse|arch [PROYECTO...]`, el camino anterior, construye entornos Podman y compila la cadena elegida desde los tar.gz registrados en `subir.lista`. `distros/dentro.sh` trabaja dentro del contenedor, no se invoca sobre el sistema anfitrión. Los `Containerfile` definen herramientas y dependencias; en Arch hay una lista auxiliar de debhelper. Los RPM/TGZ y logs quedan en `distros/salida/DISTRO/` y se registran para publicar después. No sube paquetes. Requiere espacio, red para las imágenes y repositorios y paquetes previos de la cadena. Ver el apéndice D; construir para otra distribución no es compilación cruzada de arquitectura.

`mem_util` es la biblioteca C interna del proyecto. `mem.h` declara gestión de memoria; `util.h`, archivos y cadenas; `pathconv.h`, conversión de rutas; `time_mem.h`, calendario y fechas. `libmem-util0` proporciona la biblioteca de ejecución y `libmem-util-dev` los encabezados y pkg-config. Se integra con `pkg-config -cflags -libs mem_util` o `PKG_CHECK_MODULES([MEM_UTIL], [mem_util])`. Las funciones y contratos están en `mem_util/README` y los encabezados.

`datetime()` convierte fechas completas a segundos epoch POSIX, conserva fracciones y usa la zona de la máquina o `TZ` si no se indicó un offset. Una hora sin fecha devuelve segundos desde medianoche, no epoch. Rechaza fechas imposibles y texto sobrante; no cambiar `TZ` concurrentemente. Las rutinas de día juliano conservan sus propias unidades. No confundirlas.

`etc/` contiene configuración distribuida: la del enlazador y un YAML histórico; no sustituye la configuración de publicación del usuario. `viejo/` conserva programas retirados, no órdenes recomendadas. `AUTHORS`, `COPYING` y las notas de licencias identifican autoría y excepciones; las rutinas de rutas tienen su licencia BSD propia.

```
make prueba
python3 pruebas/prueba-distro.py
python3 pruebas/prueba-subir-doc.py
# Dentro de mem_util ya configurado y compilado:
make -C mem_util check
# Documentacion, no publicacion:
(cd doc && latexmk -pdf summapack.tex)
python3 doc/web/generar.py
python3 doc/web/verificar.py
```

`pruebas/prueba-lib.sh` ejercita makefile.lib con `pruebas/ejemplo`; `prueba-bin.sh` comprueba las herramientas con destinos locales; `prueba-distro.py` simula sistemas; `prueba-subir-doc.py` valida la publicación local. No son una garantía de compatibilidad con cualquier equipo. La verificación web comprueba títulos, enlaces, tablas, presencia de componentes y copias de PDF; no ejecuta las órdenes de instalación del manual.

`doc/web/generar.py` usa Python 3 y Pandoc: convierte este manual y los documentos históricos en fragmentos HTML para TinyMCE y vistas previas. Las fuentes editables son el `.tex`, `inicio.html`, `articulo.html` y los documentos originales. `generado/` y los PDF derivados no se versionan. Se pega el fragmento en *Código fuente* del editor, no en el área visual; crear la página, guardar su contenido y subir los PDF son operaciones separadas.

17.11. makefile.lib y ubicación de sus archivos

makefile.lib. La biblioteca de make que incluyen los `Makefile` de los proyectos: unos treinta objetivos documentados (`install`, `dist`, `tgzbuild`, `ayuda`, ...). `make ayuda` lista los de cualquier proyecto que la incluya; `make prueba` la prueba con un proyecto de ejemplo y distingue lo que anda, lo que falla y los problemas conocidos. Su estado está en el apéndice O.

diccionariodepaquetes. Los nombres de los paquetes en cada distribución.

bin/. `distro` dice el sistema (UBUNTU, DEBIAN, FEDORA, SUSE, ..., y con `+` nombre y versión de `/etc/os-release`); `packinstall` es el Babel del artículo: traduce nombres con `diccionariodepaquetes` e instala con la herramienta de cada sistema, con `-prueba` para ver las órdenes sin correrlas; `summapack` da la versión instalada; `icon_install` pone lanzadores en el menú y el escritorio; `findbinlib` encuentra instalaciones viejas que taparían a las de un paquete (lo usa `instalar-tgz`). Los que nadie usaba o no andaban se pasaron a `viejo/bin`. el apéndice B tiene el relevamiento de cada uno y `pruebas/prueba-bin.sh` los prueba sin cambiar la máquina.

Dónde están. Una sola copia, en summapack:

Archivo	En el árbol	Instalado
`armar-proyecto`, `hacer-pkg.inc`, `summa.sh`, `diccionariodepaquetes`	`share/summapack/data/`	`/usr/sh`
`subir-paquetes`, `bajar-paquetes`, `instalar-tgz`	`bin/`	`/usr/bi`

`sp paquete` la toma del árbol de summapack o de summapack instalado, para los proyectos de siru y de calcula por igual. Como decía el artículo, "la inexistencia de Sumapack en una computadora que instala un paquete preparado para Sumapack no bloquea las funciones esenciales": los paquetes generados no dependen de summapack para instalarse.

17.12. Las demás herramientas de bin/

Las usa `sp`; no son órdenes para el usuario.

Herramienta	Qué hace, y quién la usa
`sp-subir`	lo que hacen `sp paquete`, `sp subir` y `sp estable`
`sp-fuentes`	las fuentes por formato (`sp fuentes`)
`sp-manifiesto`	el `MANIFIESTO`: generarlo, sincronizar svn con él, comprobar que cubra el árbol, copiar lo que dice (`sp manifiesto`, `sp paquete`)
`sp-version`	la versión completa, `AÑO.MAYOR.MENOR.rREV-N`, calculada en la copia de trabajo y escrita en la copia
`sp-instalados`, `summapack-install`	la lista de lo instalado desde fuentes (`PREFIJO/var/lib/sp/instalados/NOMBRE.json`, con la huella de cada archivo), para desinstalar sin las fuentes; `summapack-install` es `install(1)` que anota
`sp-comprobar-repo`	los mínimos que un proyecto pide de otros, contra los `.summa` estables (antes de `sp estable`)
`sp-bootstrap.in`	la parte de arriba del `sp` autónomo, que arma `distros/armar-sp`
`hacer-pkg-minimo`	el paquete mínimo, a mano y sin herramientas de construcción (sección 8)
`crear-fuentes`	el `tar.gz` de fuentes desde el `.summa`, sin compilar
`instalar-del-summa`	instala o desinstala lo que el `.summa` dice que lleva el paquete, sin herramientas de Debian
`manifiesto-instalacion`	registra y revierte los archivos de una instalación manual de summapack
`publicar-cadena`	publica la cadena de Simplot, del camino anterior de una sola base (Ubuntu 22.04)

18. Proyectos

Proyecto	Paquetes	Versión de arriba
siru/summapack	summapack, libmem-util0, libmem-util-dev	2026.2.1
siru/unidades	libunidades0, libunidades-dev	2026.1.0
siru/exprtk	libexprtk-dev	2025.01.01
siru/slex	libslex0, libslex-dev, d2datdinc	2026.1.0
siru/tabular2	libtabular2-0, libtabular2-dev, tabular	2026.2.0
siru/simplot	simplot	2026.0.2
siru/siru	siru	2026.1.0
calcula/calcula-6.0	calcula	2026.6.0
calcula/puyuspa-7.0	puyuspa	2026.7.0
siru/lanzador	lanzador (antes nuevosimusolsceptre)	2026.1.1
siru/sceptresimusol	sceptresimusol	2026.131
siru/simusol	simusol	2026.0.1
siru/dia-simusol	dia-simusol	2026.0.1
siru/Radia	radia	2026.1.2
siru/dibuja	dibuja	0.1.0

La versión completa agrega la revisión de svn y la del empaquetado (`2026.0.1.r8936-1`). Los catorce primeros van en la ISO de Ututo 2027, en el orden de `siru/ututo/proyectos-simusol.list`; todos se arman con `sp paquete`. El inventario de proyectos y paquetes está en el apéndice I.

19. Flujo de trabajo

```
# cambiar algo del paquete: editar el .summa (o el MANIFIESTO) y comitear
vi tabular2/tabular2.summa
sp manifiesto tabular2/tabular2       # si se agregaron o quitaron archivos
svn commit -m '...' tabular2/tabular2.summa tabular2/MANIFIESTO
sp paquete tabular2/tabular2          # tar.gz, .summa, paquetes y md5
sp binarios tabular2/tabular2 ututo2027
sp subir --prueba tabular2/tabular2
sp subir tabular2/tabular2
sp estable tabular2/tabular2
```

Para un proyecto nuevo: escribir su `.summa` (o copiar el de uno parecido, o `summa.sh crear` desde un `debian/`), generar el `MANIFIESTO` con `sp manifiesto`, y agregar los dos a subversion. No hace falta nada más: `sp paquete` ya lo arma.

20. Lo que falta: hoja de ruta

Propuesta (manifiesto, Rosetta, artículo de 2011)	Hoy
Rosetta: el nombre de cada paquete en cada distribución	`diccionariodepaquetes`, usado al instalar, en el rpm y en pacman; se completa a medida que un Podman limpio muestra lo que falta
`.spec`, `.lsm`, `.ebuild` desde el archivo común	`debian/`, `.spec` y el paquete de pacman; faltan `.lsm` y `.ebuild`
rpm propio de cada distribución	hecho: `sp paquete` en Fedora y SUSE, y `sp binarios`
`lacks` y `make prepare`: instalar lo que falta para compilar	hecho: el aviso de `sp paquete` con la orden, y `sp` al instalar desde fuentes
Uinstall: instalar un tgz llevando registro para desinstalarlo	hecho: `instalar-tgz`, y los desinstaladores de fuentes y tgz
Bootstrap: un script que instala summapack y sus dependencias	hecho: el `sp` autónomo, que lleva summapack adentro
Que la distribución no sea dueña del empaquetado	hecho: el `.summa`, sin repositorio, y Ututo 2027 con `sp` como gestor

Pendiente en noviembre de 2026: que cada Podman construya desde el `.dsc` o el `.src.rpm`, y no desde las fuentes publicables; una orden de `sp` para subir la ISO; y la subida al servidor del lanzamiento de noviembre, que decide el autor. Los pendientes de cada capa están en el `README`, "LO QUE TODAVIA NO CUMPLE".

A. Historia de los cambios

Cada entrada del `changelog` de summapack, y dónde está contada en este manual. El `changelog` tiene el detalle y lo medido.

Fecha	Cambio	Sección
18/11/2026	`sp iso` no se corta por cambios sin comitear ni por archivos sin declarar	12
17/11/2026	Todo documento en LaTeX se instala como PDF y como HTML (make4ht), con la configuración y el CSS comunes	14.2
17/11/2026	Los `.md` de summapack, integrados a este manual; el paquete `summapack-doc` lo instala como PDF	A y siguientes
17/11/2026	`sp paquete` arma cualquier proyecto desde su `.summa`, en el formato de la máquina; sin `hacer-pkg`	8
17/11/2026	`sp iso ututo2027`: la ISO desde nuestras fuentes; la ISO trae curl	12
17/11/2026	summapack depende de curl, y lo dice si falta	11
17/11/2026	`sp estable` pide `LUGAR/NOMBRE`	10
16-17/11/2026	La documentación se instala hecha	14.2
16/11/2026	`sp subir -tipo distro`	10
16/11/2026	Binarios por distribución: `sp binarios`, la marca, Ututo 2027	9
16/11/2026	Qué se publica y cómo se instala; el apéndice N	1, 4
16/11/2026	Las tres capas, `sp fuentes`, la comprobación del `MANIFIESTO`	5
13-15/11/2026	El `.summa` sólo con Bash (`summa.sh`); rpm y tgz directos; claves con subdirectorios	7
14/11/2026	Desinstaladores de tgz y de fuentes	11
14-15/11/2026	Imágenes de Trisquel y LFS; prueba de los generadores; Radia	9, 18
10/11/2026	lapack, blas y umfpack en el diccionario; `MANIFIESTO.excluidos`; summapack en shell	2, 5
05-09/11/2026	La capacidad Ajolote; los manuales en `doc/simusol`; lanzador, simusol y dia-simusol	13, 14, 18
03/11/2026	`sp -para fuentes -prefijo DIR`	11
02/11/2026	El `sp` autónomo lleva summapack adentro e instala sin red	11
02/11/2026	`sp` sin orden, desde el árbol, instala o actualiza summapack; el paquete mínimo	11, 8
17/10/2026	El arranque público es `sp`; binarios en distribuciones viejas	1, 9
15/10/2026	`sp` como interfaz común; los packlets	2, 11
09/10/2026	La guía `0-README-como-instalar` en la raíz de descargas	1, 4

Los apéndices que siguen son los documentos de summapack que hasta el 17 de noviembre de 2026 eran archivos `.md` sueltos en el árbol: se integraron completos a este manual, que pasó a ser su única fuente (pedido del autor). Las referencias entre ellos a esos archivos remiten ahora a su apéndice.

B. bin/ de summapack: relevamiento (07/10/2026)

B.1. Índice vigente de todos los componentes

La referencia completa de entradas, opciones, efectos, requisitos y límites está en `doc/summapack.tex`, sección «Referencia de todos los componentes», y se convierte al manual web. Esta tabla incluye también lo que no vive en `bin/`; el relevamiento de más abajo conserva la historia.

Componentes	Fuente y función	Referencia
`summapack`, `distro`	`bin/summapack.sh`, `bin/distro.sh`: versión, rutas, sistema y familia	Manual y sección distro de este archivo
`installsumma`, `bajar-paquetes`	`bin/`: arranque, descarga, selección, dependencias e instalación binaria o desde fuentes	`PAQUETES.md`
`instalar-tgz`	`bin/`: instalación con registro, respaldo, actualización y retirada	Manual
`preparar`, `fuentesexternas`	`share/summapack/data/`: dependencias de construcción y recetas externas; preparar se copia a las fuentes	Manual y `PAQUETES.md`
`packinstall`, `diccionariodepaquetes`	`bin/packinstall.pl` y `share/summapack/data/`: traducción e instalación mediante gestores del sistema	Manual
`.summa`, `summa.sh`	Descripción del proyecto y generación/consulta de metadatos	`PAQUETES.md` y ayuda de summa.sh
`hacer-pkg`, `hacer-pkg.inc`	Raíz del proyecto y datos comunes: construcción de DEB/RPM/TGZ/fuentes	`PAQUETES.md`

Componentes	Fuente y función	Referencia
`hacer-pkg-minimo`	Último recurso: arma A MANO, sin las herramientas de construcción de la distribución, los paquetes de los dos componentes `--summapack` y la biblioteca C `mem_util`, partida en `libmem-util0` y `libmem-util-dev`, que compila a mano si hay compilador y si no avisa y sigue-- en los **tres** formatos: `.deb` (el `ar` escrito ahí mismo, o `dpkg-deb` si está), `.rpm` (`armar-rpm.sh`, sin `rpmbuild`) y `.tgz`. Con `--instalar` instala el que entienda el gestor de la máquina: `dpkg`/`apt` el deb, `rpm` el rpm, ninguno de los dos el tgz con `instalar-tgz`; sin descompresores no hay paquete posible y `--forma directo` pone los archivos en su lugar, como `make install`. `--forma` arma uno solo. El recorrido completo, en `doc/instalar-sin-red-ni-herramientas.md`	`README`, `sp`
`sp` publico, `armar-sp`	`bin/sp-bootstrap.in` + `bin/bajar-paquetes` + el `.tgz` de `summapack` en base64, que arma `distros/armar-sp`: el unico ejecutable de Descargas. Con red baja e instala; **con `--sin-red`, o si la descarga falla, instala el paquete que trae adentro**: saca el tgz del propio archivo y, con `hacer-pkg-minimo --desde-tgz`, arma el `.deb` o el `.rpm` desde el y lo instala con el gestor de la maquina, o instala el tgz derecho si no hay ni `dpkg` ni `rpm`. `--raiz DIR` instala en otra raiz. `armar-sp --tgz A` incorpora otro tgz y `--sin-tgz` deja el `sp` de antes, que solo sirve con red	`doc/instalar-sin-red-ni-herramientas.m`

Componentes	Fuente y función	Referencia
`share/summapack/data/armar-rpm.sh`	Escribe un `.rpm` binario a mano (lead, cabecera de firma, cabecera con región, carga cpio+gzip), sin `rpmbuild`; `--leer` muestra los metadatos con `rpm -qip`. La disposicion de las cuatro partes, en `doc/instalar-sin-red-ni-herramientas.md`	`hacer-pkg-minimo`
`makefile.lib`	Datos comunes: reglas de construcción, instalación y dependencias	`MAKEFILE-LIB.md`
`distros/compilar`, `dentro.sh` y entornos	`distros/`: paquetes construidos dentro de Podman	`distros/README.md`
`subir-paquetes`, `subir-doc`	`bin/` y raíz: publicadores independientes de software y PDF	`PAQUETES.md` y manual
`sincronizar-pagina SISTEMA [--prueba]`	Raíz y motores Python de `share/summapack/data/`; todos los datos en `paginas-web.json`. Consulta antes de regenerar, decide la dirección y detiene conflictos. `subir-pagina` queda para compatibilidad	`doc/SINCRONIZAR-WEB.md`
`icon_install`, `findbinlib`	`bin/`: lanzadores y detección de instalaciones manuales superpuestas	Manual; revisar permisos y el modo BORRA
`templatesumma`	`bin/templatesumma.pl`: generación heredada con Template Toolkit	Su `--man` y manual; puede sobrescribir archivos
`mem_util`	Biblioteca C y paquetes de ejecución/desarrollo	`mem_util/README` y encabezados
`etc/`, `viejo/` y licencias	Configuración distribuida, programas retirados, autoría y excepciones	Manual, AUTHORS y COPYING
`doc/web/`, `pruebas/`	Generación TinyMCE, verificación y pruebas locales	`doc/web/README.md` y manual

B.2. Actualización de distro, versión 4

Se conserva `distro` sin argumentos y `distro +` (por ejemplo `UBUNTU_noble_24_04`). La detección usa `uname -s`, no el nombre del equipo ni el texto completo de `uname -a`. En Linux manda `ID` de `/etc/os-release`; si no se reconoce, se recorre `ID_LIKE` en su orden. `/usr/lib/os-release` solo se consulta si falta el primero, sin mezclar ambos archivos, conforme a la especificación de os-release (`https://www.freedesktop.org/software/systemd/man/latest/os-release.html`).

Cambio	Comportamiento
Lectura segura	Los archivos se leen como datos: no se ejecutan con `source` ni `eval`. Admite comillas, escapes, CRLF y claves repetidas (gana la última).
Familias de paquetes	openSUSE/SLES → `SUSE`; Manjaro/Arch ARM → `ARCH`; Rocky/AlmaLinux/CentOS → `REDHAT`. Son identificadores para las herramientas, no garantías de compatibilidad binaria.
WSL y contenedores	Se identifica la distribución del espacio de usuario; no se supone Ubuntu ni se toma la distribución del kernel anfitrión.
Sistemas sin ID moderno	Consulta `lsb-release` y marcadores históricos. No ejecuta tampoco `lsb-release` ni `profile`.
macOS, BSD y Solaris	Identificadores estables: `MACOSX`, `FREEBSD`, `OPENBSD`, `SUNOS`, etc.; no se mezclan con los datos Linux. macOS con `+` usa `sw_vers` si existe.
Desconocidos	`OTHERGNULINUX` para Linux sin familia reconocida y `UNKNOWN` para otro kernel desconocido.
Argumentos	Solo `+`, `--help`/`-h` y `--version`; los inválidos devuelven 2 por stderr.
Pruebas	`DISTRO_ROOT=/ruta` permite consultar archivos de otra raíz manteniendo el kernel actual; no es un chroot ni un aislamiento de seguridad. `python3 pruebas/prueba-distro.py` simula también el kernel. Integrado en la batería de binarios y en las pruebas del paquete.

El relevamiento histórico que sigue describe el estado anterior, no estas correcciones.

Actualización de nombres: el constructor se llama ahora `hacer-pkg` y su biblioteca común `hacer-pkg.inc`. Están en la raíz de cada proyecto y en `share/summapack/data/`, respectivamente, no en `bin/`. Los nombres `hacer-deb` citados en el relevamiento histórico de abajo se conservan como enlaces de compatibilidad. Se mantiene una única implementación para DEB, RPM, TGZ y fuentes `.tar.gz`.

Pedido del autor: ver como se mejora el software de summapack. Antes de cambiar nada, que hace cada script de bin/, quien lo llama, si anda y que conviene hacer con el. Se leyeron todos; se corrieron solo los que no cambian la maquina (distro.sh, y `perl -c` de los .pl). makefile.lib los instala sin la extension (distro.sh -> /usr/bin/distro).

Quien los llama: makefile.lib llama a `summapack`, `download`, `distro`, `templatesumma`, `icon_install` e `installsumma`; los Makefile de siru/make y siru/sit llaman a `packinstall.pl`. Los demas no aparecen llamados en ningun proyecto de siru ni de calcula.

script	que hace	quien lo llama	estado	propuesta
`subir-paquetes`	publica en el servidor, por proyecto, con md5 (ver PAQUETES.md)	a mano, despues de hacer-deb	nuevo, probado en local	conservar
`subir-doc` (fuente en la raíz)	publica los dos PDF de SummaPack en downloads/summapack/docs/, con respaldo y --prueba	a mano; instalado en /usr/bin/subir-doc, admite --doc DIRECTORIO	probado en local y publicado por SSH	conservar
`distro.sh`	dice la distribucion (UBUNTU, DEBIAN, SUSE, REDHAT, GENTOO, UTUTO...); con +, distribucion, nombre y version	makefile.lib	anda; la version la saca de /etc/-debian_version (en Ubuntu 24.04 dice `trixie_sid`); `--help` dice solo "AYUDA"	conservar; leer /etc/os-release; ayuda de verdad; es la base de lo que sigue
`packinstall.pl`	**Babel**: traduce nombres de paquete Debian con diccionario-depaquetes a los de cada distribucion (y modulos de Perl via cpanm) y los instala con apt, zypper, yum, urpmi, emerge	siru/make y siru/sit (`sudo packinstall.pl $(DISTRO) ...`)	anda (sintaxis bien); peligroso: apt-get con `--allow-remove-essential`, `--allow-downgrades`, agrega un PPA de toolchain, puede instalar sudo y agregar el usuario con `su`	conservar y arreglar: sacar esas opciones, `sudo` a prueba de solo muestre, que tome los nombres de BUILDDEPENDS del .summa
`download.sh`	baja un archivo de simusol.org (o de otro servidor) a un cache, comprobando el md5; la contraparte de subir-paquetes	makefile.lib	anda a medias; `--no-check-certificate` y resuelve la IP de simusol.org preguntando a 8.8.8.8 (por las jaulas de openSUSE)	rehacer como bajar-paquetes: bajar lo estable, verificar el .md5, con https verificado
`installsumma.sh`	bootstrap: baja summapack-last.tar.gz de simusol.org y hace `make install`	makefile.lib (installsumma-packMAKE)	baja con `--no-check-certificate` por IP; `--version` dice "distro.sh"	rehacer sobre bajar-paquetes: bajar el .deb o .tgz de summapack y instalarlo

script	que hace	quien lo llama	estado	propuesta
`templatesumma.pl`	arma .changes y .spec desde plantillas (Template Toolkit) con datos del .summa	makefile.lib (rpm viejo)	sintaxis bien; reemplazado por summa.sh + hacer-pkg.inc	retirar a viejo/ cuando makefile.lib use hacer-pkg.inc
`readsumma.sh`	imprime campos del .summa como asignaciones PHP/listas	nadie	campos viejos (PRIMATYSITE, mal escrito); `summa.sh campo` lo reemplaza	retirar a viejo/
`summapack.sh`	`--version` de summapack	makefile.lib (comprobar que este instalado)	dice "summapack 1.0", 2005	dar la version real del paquete (dpkg-query o el .summa)
`icon_install.sh`	instala lanzadores .desktop en el escritorio y en el lanzador de Unity	makefile.lib	Unity ya no existe (Ubuntu >= 18.04); para no root pone USRDIR=/root	simplificar a xdg-desktop-icon/xdg-desktop-menu, sin Unity
`installuserfiles.sh`	copia lanzadores y ejemplos de un paquete al usuario	nadie	errores: `$PACK_ejemplos` es una variable vacia (copia a /home/USUARIO/), el `sed` con `$PREFIX` entre comillas simples no reemplaza	unir con icon_install o retirar
`postinst.sh`	compila un paquete al instalarlo (Simusol, gfortran) y hace lo de icon_install	los postinst viejos de simusol	de la epoca de compilar en la maquina del usuario; ningun paquete actual lo usa	retirar a viejo/

script	que hace	quien lo llama	estado	propuesta
`instalartar.sh`	instala un tar de fuentes (prepare, make, make install)	nadie	no anda nunca: sale si el temporal *existe* (condicion al reves) y la comprobacion de ARCHIVOCARACTERISTICO no tiene `$`, asi que siempre dice "ya esta instalado"	retirar; lo que queria hacer (Uinstall) va como instalador de .tgz con registro
`installpkg.sh`	instalar un paquete segun la distribucion	nadie	no anda: es texto de Makefile pegado (`$$`, \ al final de lineas), borra /usr/bin/gfortran, agrega un PPA	retirar (lo hace packinstall.pl)
`findbinlib.sh`	busca (y con BORRA mueve a /tmp) programas, bibliotecas y modulos de Perl de un paquete instalados a mano en /usr/local o /home/*/bin, que taparian al del paquete	nadie en los proyectos actuales	anda; util antes de instalar un .deb sobre una instalacion vieja	conservar con ayuda; usarlo desde el instalador de .tgz
`finddir.sh`	imprime el directorio real de un script (resuelve enlaces)	nadie	el README ya dice que no esta operativo	retirar a viejo/
`finddir2.pl`	lo mismo en Perl	nadie	igual	retirar a viejo/

B.3. Lo que sale de esto

1. Retirar a `viejo/` lo que nadie llama o no anda: readsumma, postinst, instalartar, installpkg, finddir, finddir2, installuserfiles; templatesumma cuando makefile.lib use hacerdeb.inc.
2. Arreglar lo que se usa: distro (os-release, ayuda), packinstall (sin opciones peligrosas, `--prueba`), summapack (version real), icon_install (sin Unity).
3. Lo nuevo que falta, segun la hoja de ruta de PAQUETES.md:

    - `bajar-paquetes`: la contraparte de subir-paquetes (reemplaza download e installsumma), con md5 y https verificado.

- instalar un .tgz con registro, para poder desinstalarlo (Uinstall del articulo de 2011), usando findbinlib para avisar lo que tapa.
- `make prepare` con packinstall y el BUILDDEPENDS del .summa.

4. Cada script que queda: `--help` que explique, `set -eu`, y una prueba en `pruebas/` como la de makefile.lib.

B.4. Hecho (07/10/2026, pasos 1 y 2)

- A `viejo/bin/`: readsumma, postinst, instalartar, installpkg, finddir, finddir2, installuserfiles. templatesumma queda hasta que makefile.lib use hacer-deb.inc.
- `distro`: version y nombre de /etc/os-release (UBUNTU_noble_24_04, antes UBUNTU_DEBIAN_trixie_sid); Fedora sale FEDORA (antes REDHAT, porque tambien tiene /etc/redhat-release); un sistema no reconocido sale OTHERGNULINUX (antes SUSE: faltaba un `$`); la rama de macOS nunca entraba (faltaba una `x`); `--help` explica.
- `packinstall`: `--prueba` muestra las ordenes sin correrlas; apt-get sin `--allow-remove-essential --allow-downgrades --allow-change-held-packages`; ya no instala sudo ni agrega usuarios al grupo sudo (dice como hacerlo y sale); sin DISTRO la deduce y la usa (antes el primer paquete quedaba como distro); un REPO-ppa se busca por su nombre (antes se miraba siempre el de toolchain); `~/.cpanm` se comprueba de verdad; se saco fortran6, que borraba /usr/bin/gfortran (nunca se llamaba).
- `summapack`: dice la version instalada (dpkg o rpm) y donde estan sus datos; `--help` lista las herramientas.
- `icon_install`: sin Unity; menu con xdg-desktop-menu y escritorio con xdg-user-dir; `--prueba`; para un usuario comun ya no usa /root.
- `pruebas/prueba-bin.sh` (en `make prueba`): 26 comprobaciones, sin cambiar la maquina (distro con un /etc de mentira, el resto con --prueba).

B.5. Hecho (07/10/2026, pasos 3 y 4)

- `bajar-paquetes` (nuevo): la contraparte de subir-paquetes. Baja lo estable (o `--inestable`, o `--version V`) en el formato del sistema (deb, rpm o tgz; o `--formato`), con su .md5, por https con certificado; si el md5 no coincide lo descarta. Lo deja con su nombre de verdad y con `--instalar` lo instala (apt, dnf/zypper, instalar-tgz). `--prueba`.
- `instalar-tgz` (nuevo): el Uinstall del articulo. Instala un .tgz llevando registro en /var/lib/summapack/tgz (`--lista`, `--archivos`, `--quitar`); no pisa archivos de paquetes dpkg o rpm (salvo `--forzar`); respalda lo que pisa y lo devuelve al quitar; al actualizar saca lo que la version nueva ya no trae; avisa con findbinlib lo que lo taparia en el PATH. `--raiz` para instalar en otro arbol, `--prueba`.
- `installsumma` (el bootstrap): baja bajar-paquetes de la raiz del servidor e instala summapack y mem_util (despues, 07/10/2026: tambien los proyectos que se le pidan). `sp subir --bootstrap summapack` (en summapack) los sube; `make installsummapackMAKE` (makefile.lib) los usa, con curl y https, sin la IP por 8.8.8.8 ni `--no-check-certificate`.
- `download.sh` a viejo/bin: nadie lo llamaba; lo reemplaza bajar-paquetes.
- Dependencias de compilacion: `summa.sh dependencias DIR` da los nombres de BUILDDEPENDS; hacer-deb las comprueba antes de compilar y, si faltan, dice la orden para instalarlas (packinstall o apt-get); `make dependencias` (makefile.lib) las instala con packinstall (`PRUEBA=1` solo muestra).
- `findbinlib`: `--help`; /bin ya no cuenta como otro lugar en los sistemas con /usr unificado (marcaba /bin/ls como si tapara a /usr/bin/ls).
- `pruebas/prueba-bin.sh`: 39 comprobaciones, con un servidor y una raiz de mentira para subir, bajar, instalar, actualizar y quitar.

Queda: templatesumma (hasta que makefile.lib arme los paquetes con hacer-deb.inc), y que makefile.lib use hacer-deb.inc para `make deb/rpm`.

Despues (07/10/2026, pedidos del autor): el servidor de descargas quedo organizado por proyecto y sin nombres `_last`; bajar-paquetes baja primero PROYECTO.summa y con el los binarios. Ver PAQUETES.md.

C. Entornos de construcción y prueba

Las diez raíces mínimas se conservan como `salida/entornos-limpios/<entorno>.tar.gz`; `preparar-diez-entornos` crea las imágenes Podman correspondientes y `exportar-diez-entornos` produce los archivos. Las variantes con herramientas de desarrollo deben ser imágenes distintas para no alterar estas copias limpias. Las cifras de glibc se midieron dentro de las imágenes el 15/10/2026. GCC no viene instalado en las raíces mínimas; esta tabla se completará con el GCC efectivamente instalado, no con el supuesto por el nombre de la distribución.

C.1. Protección durante las pruebas

Elemento	Regla	Recuperación
Fuentes y scripts del árbol	Se montan en Podman con `:ro`; `distros/compilar` también monta el archivo de fuentes en solo lectura y descomprime en un directorio temporal del contenedor.	Los cambios voluntarios del árbol se controlan con SVN; las pruebas no deben escribir allí.
Imagen `:base`	Las pruebas usan `podman run --rm`, que descarta su capa escribible al terminar. `sp podman` no vuelve a descargar ni reemplaza una base existente.	La imagen base permanece disponible para repetir la prueba.
Imagen `:desarrollo`	`preparar-desarrollo` no reemplaza una existente; valida la candidata antes de etiquetarla.	Si una prueba falla, se repite desde `:base`; la imagen anterior conserva su etiqueta.
`salida/entornos-limpios/*.tar.gz`	`exportar-diez-entornos` comprueba y reutiliza cada archivo existente; nunca lo sobrescribe.	Si se desea otra exportación, se retira explícitamente la anterior después de conservarla por el medio elegido.

No se crean copias de seguridad de cada ejecución: la reversibilidad normal proviene del árbol SVN y de las capas inmutables de Podman. Esta protección cubre las herramientas citadas, no cambios manuales hechos fuera de ellas. En `arch-viejo` solamente, la imagen de desarrollo usa el archivo histórico oficial por HTTPS con comprobación de firmas desactivada para esos repositorios, porque las claves de 2022 están vencidas; esto no modifica el anfitrión ni la imagen base.

Entorno	Imagen construida	glibc medida	GCC en la raíz mínima	Candidata para igualar las bases antiguas en 2.35
Ubuntu antiguo	22.04	2.35	No	Mantener
Debian antiguo	12	2.36	No	Debian testing de octubre de 2022; no es una versión estable publicada
Fedora antiguo	36	2.35	No	Mantener
openSUSE antiguo	Leap 15.4	2.31	No	Tumbleweed de antes de fines de agosto de 2022
Arch antiguo	Imagen del 31/05/2022	2.35	No	Mantener
Ubuntu actual	26.04	2.43	No	No cambiar
Debian actual	13	2.41	No	No cambiar
Fedora actual	44	2.43	No	No cambiar
openSUSE actual	Tumbleweed	2.44	No	No cambiar
Arch actual	Imagen actual	2.44	No	No cambiar

La propuesta Debian testing está respaldada por el historial de glibc de Debian (`https://tracker.debian.org/pkg/glibc/news/?page=7`), que registra la migración de 2.35 a testing en septiembre de 2022. Debian 12 estable (`https://packages.debian.org/en/bookworm/libc6`) usa 2.36. openSUSE (`https://news.opensuse.org/2022/09/02/tw-continues-release-streak/`) informa que Tumbleweed pasó a glibc 2.36 a fines de agosto de 2022. Esas dos sustituciones son candidatas, no imágenes creadas.

D. siru/summapack/distros: compilar en otras distribuciones

Pedido del autor, 07/10/2026: "hagamos entornos para compilar los paquetes en esos entornos cada vez que tengamos estos problemas, asi proveemos paquetes para suse actual, fedora actual y arch actual". El problema esta en siru/summapack/PAQUETES.md ("Otras distribuciones"): lo compilado en Ubuntu pide su libc y sus versiones de bibliotecas, y no siempre anda en otra.

D.1. Uso

Un proyecto se arma en el formato de la máquina con `sp paquete LUGAR/NOMBRE`, y en el Podman de cada distribución con `sp binarios LUGAR/NOMBRE DISTRO...`. Los formatos y los requisitos de cada imagen se describen en formatos-de-paquete.md.

```
summapack/distros/compilar fedora                          toda la cadena
summapack/distros/compilar suse tabular2                   un proyecto
summapack/distros/compilar arch summapack unidades slex exprtk tabular2
```

Hace falta podman, y antes el sp paquete de cada proyecto: se compila desde el tar.gz de fuentes que deja (el de subir.lista), que trae el debian/ generado del .summa. La cadena, en orden (cada uno usa los anteriores ya compilados para esa distribucion): summapack, unidades, slex, exprtk, tabular2, siru, calcula, puyuspa.

D.1.1. Diez entornos mínimos y sp

`preparar-diez-entornos` llama diez veces a `sp podman ENTORNO`; el catálogo de imágenes por distribución y época está en `sp/packlets/podman.sh`. Cada llamada crea la etiqueta `localhost/simusol-<entorno>:base`. También se puede ejecutar una sola variante, por ejemplo `sp/sp podman suse-viejo`, o inspeccionar sus órdenes sin descargarlas con `sp/sp --ensayo podman suse-viejo`. `exportar-diez-entornos` conserva cada raíz limpia como `salida/entornos-limpios/<entorno>.tar.gz`. Las versiones medidas de glibc y GCC están en `ENTORNOS.md`. Las imágenes de desarrollo son copias separadas: nunca se instalan compiladores en los archivos limpios.

Orden	Efecto
`bash distros/comprobar-repos-desarrollo ENTORNO`	Monta el árbol de SummaPack en solo lectura y ejecuta `sp buildtools` en un contenedor descartable. Comprueba la instalación real sin modificar la imagen base ni el anfitrión.
`bash distros/preparar-desarrollo ENTORNO`	Ejecuta el mismo `sp buildtools` en una copia de la base, verifica compiladores, GSL, OpenBLAS, SuiteSparse y glibc, y solo entonces etiqueta la imagen como `:desarrollo`; si ya existe, no la reemplaza.

Las órdenes se ejecutan desde `siru/summapack`. Si se omite `ENTORNO`, cada script recorre su lista predeterminada. `sp` detecta la distribución dentro del contenedor y su packlet ejecuta el gestor nativo: primero actualiza índice y paquetes instalados, luego instala las herramientas y bibliotecas de desarrollo. Por eso los dos scripts no contienen nombres de paquetes ni comandos propios de APT, DNF, Zypper o Pacman. La prueba de repositorios instala solo en un contenedor efímero; es más fuerte que una simulación de dependencias, pero vuelve a descargar paquetes cuando después se prepara la imagen persistente.

Arch histórico usa la raíz limpia de mayo de 2022, con glibc 2.35. Las claves de esa instantánea están vencidas; solo dentro de los contenedores de desarrollo de `arch-viejo` se usa su archivo oficial por HTTPS con firmas desactivadas para esos repositorios. La imagen base y el anfitrión conservan su configuración original. `buildtools` obtiene la lista de dependencias de `sp` y traduce sus nombres con `diccionariodepaquetes`; en Arch `libopenblas-dev` instala `openblas` y `cblas`.

Los paquetes quedan en `summapack/distros/salida/DISTRO/` (fuera de svn), con un `PROYECTO.log` por proyecto, y se agregan a `siru/subir.lista`: se suben con

```
subir-paquetes $(cat ~/svn/siru/subir.lista)
```

y se marcan estables con `subir-paquetes --estable PROYECTO`, como los de Ubuntu.

D.2. Que hay

archivo	que es
`fedora/Containerfile`	Fedora actual (fedora:latest)
`suse/Containerfile`	openSUSE Tumbleweed
`debian13/Containerfile`	Debian 13, para probar paquetes DEB construidos localmente desde fuentes
`ubuntu2404/Containerfile`	Ubuntu 24.04, para probar paquetes DEB construidos localmente desde fuentes

archivo	que es
`arch/Containerfile`, `arch/debhelper.lista`	Arch; debhelper de Ubuntu 24.04 desde sus .deb (Arch no lo tiene), con sha512
`lfs/construir`, `lfs/bootstrap`, `lfs/armar`	Compilan LFS desde fuentes con jhalfs, conservan la raíz e importan su Podman para el TGZ genérico
`trisquel/armar`, `trisquel/Containerfile`	Importan una raíz oficial Trisquel y agregan herramientas desde sus propios repositorios
`compilar`	arma el entorno y compila cada proyecto en el
`dentro.sh`	lo que corre adentro del contenedor

D.3. Como funciona

Fedora, openSUSE y Arch tienen dpkg y fakeroot, y Fedora y openSUSE tambien debhelper. Asi se compila con la misma receta que en Ubuntu (el debian/rules que sale del .summa), pero con el compilador y las bibliotecas de la distribucion:

1. Se instalan los paquetes nuestros ya compilados para esa distribucion (los de `salida/DISTRO/`) y lo que pide BUILDDEPENDS, traducido con el diccionariode-paquetes de summapack.
2. `dpkg-buildpackage` compila y corre las pruebas (las de tabular2, por ejemplo, corren adentro de cada distribucion).
3. El .deb que sale es solo un paso intermedio: de su arbol se arma el **.rpm** (Fedora, openSUSE), con las dependencias de las bibliotecas calculadas por rpm mirando los binarios de esa distribucion, o el **.tgz** (Arch). El .deb se descarta.

Los paquetes llevan la marca de la distribucion, para convivir en el servidor con los de Ubuntu; bajar-paquetes busca primero los de su distribucion elegida con `--para` y, si no hay, los generales. No lo decide `distro.sh`; un paquete general de otra distribución puede resultar incompatible:

distribucion	marca	ejemplo
Fedora (la actual, hoy 44)	fedora	`tabular-2.0-15.fedora.x86_64.rpm`
openSUSE Tumbleweed	suse	`tabular-2.0-15.suse.x86_64.rpm`
Arch	arch	`tabular_2.0-15+arch_amd64.tgz`

Hasta el 08/10/2026 las marcas eran `fc44` y `tw`; el autor pidio el nombre de la distribucion.

Politica actual: los binarios publicados se construyen en las imagenes antiguas de cada familia: Ubuntu 22.04 para `.deb`, Fedora antiguo y openSUSE antiguo para sus respectivos `.rpm`, y Arch antiguo para `.tgz`. Los tarballs de fuentes se prueban por separado en las cinco imagenes actuales (Ubuntu, Debian, Fedora, openSUSE y Arch). No se convierten los binarios de Ubuntu en RPM o TGZ para publicarlos como si fueran nativos. Las diez imagenes y sus versiones estan registradas en `ENTORNOS.md`.

Hay que recompilar seguido (lo que sigue vale para los paquetes de cada distribucion). Estos paquetes sirven para la version de la distribucion en que se compilaron. Con las bibliotecas nuevas, un paquete viejo puede dejar de andar: entonces se recompila en su imagen y se vuelve a subir. La instalacion desde fuentes (`sp --para fuentes`) construye contra las bibliotecas de la maquina destinataria.

En los entornos, algunas herramientas de debhelper que solo tienen sentido en Debian (dh_shlibdeps, dh_strip_nondeterminism, dh_dwz) son comandos que no hacen nada, y no se arman los paquetes -dbgsym.

D.4. Pruebas locales actuales (12/10/2026)

Entorno	Comprobación desde fuentes
Debian 13	`make install`/`make uninstall` de SummaPack con `DESTDIR`; ExprTk: tarball → DEB local → APT → retirar; construcción DEB de SummaPack y `libmem_util`
Ubuntu 24.04	`make install`/`make uninstall` de SummaPack con `DESTDIR`; ExprTk: tarball → DEB local → APT → retirar
Fedora 44	`make install`/`make uninstall` de SummaPack con `DESTDIR`; ExprTk: tarball → RPM local → DNF → retirar; SummaPack y `libmem_util`: tarball → RPM locales → DNF
openSUSE Tumbleweed 20261005	`make install`/`make uninstall` de SummaPack con `DESTDIR`; ExprTk: tarball → RPM local → Zypper → retirar; SummaPack y `libmem_util`: tarball → RPM locales → Zypper

Estas pruebas se hicieron dentro de contenedores efímeros; no instalaron paquetes en el host. La construcción local usa las fuentes y dependencias de cada entorno. No certifican aún toda la cadena de proyectos ni la instalación del bootstrap público desde cero.

D.5. Lo que aparecio al compilar alli

Compilar con el GCC 15 de Fedora 44 encontro errores que el de Ubuntu solo avisaba, y se corrigieron en las fuentes:

- unidades 1.0-2: `gnu-units/units.c` asignaba NULL a un caracter.
- tabular2 2.0-15: `tests/t_tablas.c` usaba una funcion sin incluir su declaracion (cell.h).

D.6. Probado (07/10/2026)

En contenedores limpios, `sh installsumma.sh tabular2` con un servidor con los paquetes de Ubuntu y los de cada distribucion: Fedora 44, openSUSE Tumbleweed, Arch y Ubuntu 24.04 bajan cada uno los suyos, instalan, y `interpola` anda (con gnuplot).

Falta: compilar siru, calcula y puyuspa en las tres.

E. Agregado al Manifiesto Anti-distros: la capacidad Ajolote

Pedido del autor, 05/11/2026, a partir del caso de `sceptresimusol`. Va como **requisito necesario de los paquetes modernos**.

El manifiesto que está en este directorio (`UTUTO-ANTIDISTROS.html`) es una captura de la página publicada —dice «Última actualización: 24 de septiembre de 2026»—, así que no se edita acá: este archivo es el texto del agregado, para incorporarlo a la página. La versión larga, con el detalle técnico, está en la documentación de summapack (`doc/summapack.tex`, sección «La capacidad Ajolote») y el caso concreto en `siru/sceptresimusol/docs/AUTOINSTALACION.md`.

E.1. El texto

Si el empaquetado es parte del desarrollo de cada programa y no de cada distribución, entonces hay algo más que el programa no puede delegar: **volver a existir cuando el sistema operativo cambia**.

Todo paquete binario muere al cambiar el sistema operativo: queda atado al compilador y a las bibliotecas con que se construyó. Lo habitual es que el sistema nuevo traiga otro paquete construido para él, pero eso supone que alguien lo va a construir y distribuir, y nada lo asegura. Para un programa mantenido por un grupo chico, suponerlo es perder el programa.

De ahí el requisito: **cada paquete binario debe traerse sus propias fuentes y poder rehacerse a sí mismo**. El binario instala el `tar` de sus fuentes y su `.summa` junto con lo demás, y el programa avisa, cuando detecta que el entorno cambió, con la orden exacta para reconstruirse:

```
sp --para /usr/share/PAQUETE/fuentes install PAQUETE
```

A esa propiedad la llamamos **capacidad Ajolote**: el ajolote se reconstruye desde casi cualquier parte, siendo un animal complejo. La *Hydra* común lo hace más —y además no muere en condiciones favorables—, pero un programa como SCEPTRE es complejo, así que se parece más a un ajolote.

Tiene un pariente conocido: **DKMS** (*Dynamic Kernel Module Support*) hace lo mismo con los controladores que no están en el núcleo oficial —instala las fuentes en `/usr/src/` y el sistema las recompila al cambiar el núcleo—, con tres diferencias: DKMS sirve sólo para módulos del núcleo, lo dispara el sistema y no la persona, y existe en las distribuciones de Linux y no fuera de ellas. La reconstrucción de acá la hace `sp`, que arma `.deb`, `.rpm` y `.tgz`.

F. Formatos de paquete

`sp paquete LUGAR/NOMBRE` arma un proyecto desde su `.summa` y su `MANIFIESTO`, en el formato de la máquina donde corre, y deja el resultado al lado del proyecto. No instala ni publica nada. Para armar en otra distribución está `sp binarios LUGAR/NOMBRE DISTRO...`, que hace lo mismo dentro del Podman de cada una.

Máquina (según `/etc/os-release`)	Paquete
Debian, Ubuntu, Trisquel, Ututo 2027	DEB, y de él un TGZ y un RPM con los mismos archivos
Fedora	RPM propio, con marca `fedora`
openSUSE	RPM propio, con marca `suse`
Arch	`pkg.tar.zst` de pacman
cualquier otra	TGZ

En todos los casos quedan además el `NOMBRE_VERSION.tar.gz` de las fuentes y el `NOMBRE_VERSION.summa`, cada uno con su `.md5`. La variable `SP_FORMATO=deb|rpm|pacman|tgz` fuerza el formato; la usan las pruebas en Podman.

Ningún proyecto trae su propia receta: lo que cambia de uno a otro está en su `.summa`. Lo que hace falta para construir es el `BUILDDEPENDS`, y un paso propio antes de armar las fuentes (generar un `.tab.c` con bison, por ejemplo) va en `FUENTES_PREPARAR` (17/11/2026; antes cada proyecto tenía un `hacer-pkg`).

F.1. Pasos comunes

1. Una copia limpia: exactamente lo que dice el `MANIFIESTO`. La versión, con `sp-version`.

2. En la copia, el `.summa` genera la capa automake (`instalar.am`, `dist.am` y, si el proyecto no compila nada, `configure.ac` y `Makefile.am`), `debian/` y el `.spec`; corre `FUENTES_PREPARAR` y queda hecho el `configure`. Esa copia es el `tar.gz` de las fuentes.
3. Se compila y se ejecuta `make install` con `DESTDIR` en un directorio temporal. En DEB lo hace `debian/rules` (`dh_auto_install`); en RPM, el `.spec`.
4. Cada paquete toma los archivos que declara su `ARCHIVO_NOMBRE_install` en `.summa` (o `ARCHIVO_install` si hay uno solo). La documentación que va a `usr/share/doc/simusol` se instala hecha: el `.html` de cada `.md` y el `.pdf` de cada `.tex` con `\documentclass`. Un RPM, un paquete pacman o un TGZ de Fedora, SUSE o Arch se construyen desde esas fuentes y esa instalación, no se convierten desde un DEB.
5. Las dependencias salen de `DEPENDS` y `NOMBRE_DEPENDS` en `.summa`. El DEB conserva los nombres de Debian; los RPM consultan las entradas `FEDORA` o `SUSE` de `diccionariodepaquetes` y completan con las entradas `RPM`; pacman consulta `ARCH`. El TGZ lleva un archivo de control con los nombres originales.

F.2. Herramientas y estado

Formato	Herramienta de armado en el entorno
DEB	`dpkg-buildpackage` y `debhelper`
RPM	`rpmbuild`
pacman	`bsdtar`, `tar` y `zstd` para el archivo y los metadatos ALPM
TGZ	`tar` y `gzip`

`sp/tests/test_generadores_podman.sh` prueba en las imágenes disponibles el paquete de summapack en cada formato y, dentro de cada entorno, un `make install` de ExprTk con su desinstalador. Acepta nombres de formas para repetir una prueba aislada, por ejemplo `sp/tests/test_generadores_podman.sh fedora suse`.

La generación de `debian/`, del `.spec` RPM y de los archivos de Autotools usa `summa-debian.sh`, `summa-rpm.sh` y `summa-automake.sh`, respectivamente. Los tres leen el `.summa` mediante Bash. GNU Autoconf y Automake siguen siendo herramientas externas para construir el proyecto; esos scripts solo generan sus archivos de entrada. El paquete mínimo (`bin/hacer-pkg-minimo`, lo usa `sp` solo cuando faltan las herramientas) conserva el constructor básico para el arranque sin herramientas de distribución; su funcionamiento se explica en `instalar-sin-red-ni-herramientas.md`.

Para un proyecto con `Makefile.am` propio, el generador incorpora `instalar.am` si todavía no está incluido. La opción `summa-automake.sh --solo-instalar DIRECTORIO` regenera únicamente esa parte y conserva el resto de las reglas del proyecto; SCEPTRE la usa porque mantiene un `dist.am` propio.

F.3. Desinstalar instalaciones directas

Instalación	Script que queda instalado	Registro que utiliza
TGZ mediante `instalar-tgz`	`/var/lib/summapack/desinstalar/tgz/NOMBRE.sh` (o bajo la raíz pasada con `--raiz`)	`/var/lib/summapack/tgz/NOMBRE.lista` y respaldo de archivos reemplazados
Fuentes mediante `sp --para fuentes`, que ejecuta `make install`	`PREFIX/var/lib/summapack/desinstalar/fuentes/NOMBRE.sh`	`PREFIX/var/lib/summapack/fuentes/NOMBRE.js`

Instalación	Script que queda instalado	Registro que utiliza
Fuentes mediante `make install` directo en un proyecto generado por `summa-automake.sh`	`PREFIX/var/lib/summapack/desinstalar/fuentes/NOMBRE.sh`	`PREFIX/var/lib/summapack/sp-instalados/NOMBRE.js`

Los scripts ejecutan los desinstaladores ya existentes y se eliminan al quitar el paquete. Con el prefijo habitual `/usr/local`, el de fuentes queda en `/usr/local/var/lib/summapack/desinstalar/fuentes/`. El objetivo `install-data-hook` generado por `summa-automake.sh` registra también un `make install` directo, sin necesidad de conservar las fuentes para desinstalar. Para conocer todos los archivos, incluidos los instalados por subdirectorios y reglas propias, repite la instalación en una raíz temporal con `DESTDIR` y registra los archivos que quedaron también en el prefijo real. Crea el `.sh` si la versión instalada de `sp-instalados` todavía no lo hace. Cuando se instala en `DESTDIR` para construir un paquete, omite ese registro. `make uninstall` quita también el registro de fuentes.

F.4. Imagen de Linux From Scratch

`distros/lfs/construir` descarga las fuentes y compila LFS 13.1 con jhalfs, la automatización oficial (`https://www.linuxfromscratch.org/alfs/`), en un Podman privilegiado y descartable. Conserva las fuentes, registros y avances en `distros/salida/lfs`, genera `raiz-lfs.tar` y llama a `distros/lfs/armar` para importar y probar la imagen. La raíz no está incluida en SVN. Hace falta una terminal de al menos 80 columnas y 24 líneas, conexión para descargar las fuentes y espacio para compilar el sistema. Si ya se dispone de una raíz LFS, se puede importar con `distros/lfs/armar RAIZ-LFS.tar.gz`. El libro oficial (`https://www.linuxfromscratch.org/lfs/view/stable-systemd/`) describe los paquetes construidos; LFS no impone un gestor de paquetes.

F.5. Imagen de Trisquel

`distros/trisquel/armar` crea una raíz mínima de Trisquel 12 Ecne (`https://trisquel.info/en/trisquel-120-ecne-release-announcement`) con `mmdebstrap`, verifica la firma del archivo de paquetes, la importa como base y agrega las herramientas de construcción desde el repositorio oficial de Trisquel (`https://trisquel.info/en/wiki/packages-repository`). El contenedor Ubuntu que ejecuta `mmdebstrap` es solo la herramienta de creación; los archivos de la imagen resultante proceden de Trisquel. La raíz queda en `distros/salida/trisquel/raiz.tar` para repetir la importación sin descargarla otra vez. También se puede pasar una raíz ya construida: `distros/trisquel/armar RAIZ-TRISQUEL.tar.gz`. La orden comprueba su identidad y sus repositorios antes de importarla.

G. Instalar summapack sin red y sin herramientas de empaquetado

Como se instala summapack desde el arbol de fuentes en una maquina que no tiene Internet y tampoco tiene las herramientas con que se fabrican paquetes (dpkg-dev, debhelper, fakeroot, autotools, rpmbuild). Es el caso extremo: sin red no se pueden bajar esas herramientas, y sin esas herramientas no se puede armar el paquete de la manera normal. Pedido del autor, 02/11/2026.

Lo normal esta en otros documentos: con red y con el servidor, en `doc/0-README-como-instalar`; bajando los paquetes a mano, en `doc/instalar-sin-summapack.md`; el paquete normal, en `PAQUETES.md` y `BIN.md`.

G.1. Lo que hay que entender primero

Dos cosas distintas, que suelen confundirse:

- el GESTOR de paquetes --`dpkg`, `apt`, `rpm`, `dnf`, `zypper`--, que instala paquetes ya hechos. En Debian y Ubuntu `dpkg` es `Essential: yes` y esta siempre; en las distribuciones de rpm, `rpm` esta siempre.
- las HERRAMIENTAS DE CONSTRUCCION --`dpkg-dev`, `debhelper`, `fakeroot`, `rpmbuild`, autotools, compilador--, que fabrican paquetes. Esas no estan siempre, y son justo las que no se pueden traer sin red.

De ahi sale la idea de todo esto: **el paquete se escribe a mano, con lo que trae cualquier sistema, y se instala con el gestor, que ahi siempre esta.** No se puentea `dpkg` ni `rpm`: se les da de comer un paquete legitimo.

G.2. El recorrido: `sp` sin argumentos, desde el arbol

```
cd /ruta/al/arbol/summapack
sp/sp                       # ojo: sin «sh», que sp es bash
```

`sp` sin accion, corrido desde el arbol, es la autoinstalacion. Arranca si summapack no esta instalado, o si esta instalado con una version mas vieja que la del arbol (se compara con `dpkg --compare-versions` donde hay apt, y con `sort -V` en las demas). Si la instalada es igual o mas nueva, no hace nada: dice «falta la operación» y termina con 2, como cualquier `sp` sin accion.

Con `--ensayo` dice lo que haria sin tocar la maquina.

Los pasos:

1. **Herramientas.** Mira que falta de `HERRAMIENTAS_PAQUETE` (los nombres se traducen por distribucion). Si falta algo y hay red, lo instala con el gestor. **Si no falta nada no llama al gestor: no refresca, no actualiza la distribucion y no usa la red.** Si falta algo y no hay red, avisa y sigue con lo que haya en la maquina en lugar de cortar.

1b. **Ultimo recurso.** Si despues de eso falta alguna de las ordenes que `hacer-pkg` exige para el paquete normal (`svn`, `dpkg-buildpackage`, `dh`, `fakeroot`, `autoreconf`, `libtoolize`, `gcc`), se arman los paquetes minimos con `bin/hacer-pkg-minimo` y se instalan. Termina ahi.

2. **El paquete normal**, con `sp-subir --paquete-modo` (que adentro usa `hacer-pkg`), con la version del arbol.

3. **La instalacion** de lo que acaba de armar: el `.deb` donde hay apt, el `.rpm` donde hay dnf o zypper, el `.tgz` en las demas.

El paso 1 se podia trabar antes: `apt-get` sin red termina con 100 y el error subia hasta cortar `sp` antes de llegar al paso 2. Por eso ahora se mira primero que falta y solo se llama al gestor si hay algo que traer.

G.3. El ultimo recurso: `hacer-pkg-minimo`

```
hacer-pkg-minimo [--instalar] [--raiz R] [--prueba] [--forma F] [DIR]
```

Arma a mano los paquetes de los dos componentes de summapack, los mismos tres que arma `hacer-pkg`:

```
summapack         independiente de arquitectura (all/noarch): los archivos
                  que el `.summa` manda instalar, que son todos guiones y
                  datos
libmem-util0      la biblioteca C mem_util compartida
libmem-util-dev   sus encabezados, el .a y el mem_util.pc
```

y cada uno en los **tres formatos**: `.deb`, `.rpm` y `.tgz`. Son nueve archivos de una corrida. No se arma solo el que sirve en esta maquina, igual que `hacer-pkg`, porque para eso estan los formatos: los otros sirven para llevarlos a otra. Con `--forma` se arma uno solo.

Los archivos salen del mismo listado que usan el `make install` y el paquete normal: `ARCHIVO_summapack_install` del `summapack.summa`, que es shell y se lee con `.`. Como salen del arbol de trabajo y no de un `svn export`, se saca antes lo que pudo quedar pegado y no corresponde: `__pycache__`, `.svn`, `*.pyc`, `*~`, `*.orig`, `*.rej`.

G.3.1. Que se instala

Lo que entienda el gestor de la maquina, que es lo que esta aunque no esten las herramientas de construccion:

```
con dpkg o apt         el .deb, con `dpkg -i`
con rpm                el .rpm, con `rpm -U --nodeps`
con ninguno de los dos el .tgz, con `instalar-tgz`, que lleva el registro
                       de lo que puso para poder sacarlo con --quitar
sin descompresores     no se puede armar ningun paquete: queda
                       `--forma directo`
```

`--forma directo` pone los archivos en su lugar, que es lo que hace un `make install`, y los anota en el mismo registro de `instalar-tgz` para poder sacarlos despues. Hace falta porque el `make install` del arbol necesita autotools, que en este caso es justo lo que no hay.

`--raiz R` instala en `R` en lugar de en `/`: una imagen, una jaula, o para probar sin ser root. Se pide `sudo` solo si hace falta: si la raiz es escribible por el usuario, no.

G.3.2. Las dependencias no se declaran

A proposito: estos paquetes existen para arrancar una maquina donde no se puede bajar nada, asi que no pueden exigir nada. Lo que el paquete normal pone en `Depends` aca va en `Recommends`, que apt no obliga a satisfacer.

G.3.3. La version lleva `~minimo`

```
summapack_2026.2.1.r6087-1~minimo_all.deb
```

Para `dpkg --compare-versions`, `1-1~minimo` es ANTERIOR a `1-1`. Asi el paquete completo actualiza al minimo en cuanto haya con que fabricarlo, sin tener que desinstalar nada ni forzar versiones.

G.4. Sin arbol: el `sp` suelto con el paquete adentro

Todo lo de arriba sale del arbol de fuentes. Falta el caso en que no hay arbol y lo unico que hay es el `sp` suelto, el ejecutable publico de Descargas. Para eso el `sp` lleva el paquete adentro, como datos.

G.4.1. Como esta hecho ese sp

Lo arma `distros/armar-sp`, y son tres pedazos pegados:

```
bin/sp-bootstrap.in       el arranque, /bin/sh, con @TGZ@ sustituido por
                          el nombre del tgz que lleva adentro
__SP_PAYLOAD__
bin/bajar-paquetes        el descargador
__SP_TGZ__
<el .tgz de summapack en base64>
```

Cada pedazo se saca del propio archivo cuando hace falta: el descargador con un `sed` entre las dos marcas, y el tgz con `sed` desde la segunda marca mas `base64 -d`. Son unos 436 K en total, de los que 250 K son el paquete.

```
armar-sp                  incorpora el tgz ESTANDAR de la version del
                          arbol, el que fabrica hacer-pkg
                          (summapack_VERSION_all.tgz); si no esta armado,
                          avisa y pone el minimo, que arma al vuelo en un
                          directorio temporal
armar-sp --estandar       no acepta ese reemplazo: corta y dice que hay
                          que correr «sp paquete summapack». Es lo que usa
                          «sp subir --bootstrap», porque lo que se publica
                          tiene que llevar el paquete de verdad
armar-sp --tgz A.tgz      incorpora ese tgz
armar-sp --sin-tgz        el sp de antes, que solo sirve con red
```

El estandar sirve para cualquier maquina porque el paquete de summapack es el independiente de arquitectura: 103 archivos, 21 directorios, un enlace y NI UN BINARIO (los binarios estan en libmem-util0 y libmem-util-dev, que son de arquitectura y no viajan adentro del sp).

G.4.2. Como instala sin red

```
sh sp                     con red: baja e instala, como siempre
sh sp --sin-red           del paquete que trae adentro, sin tocar la red
sh sp --sin-red --raiz R  en otra raiz: una imagen, una jaula, o probar
sh sp --prueba --sin-red  decir que haria
```

Y si la descarga falla --no hay red, el servidor no esta-- avisa y pasa solo al paquete de adentro, en lugar de terminar con error.

Del tgz se saca `usr/bin/hacer-pkg-minimo`, que es parte del propio paquete, y se lo llama con `--desde-tgz`:

```
hacer-pkg-minimo --desde-tgz A.tgz [--instalar] [--raiz R] [--salida D]
```

El `.tgz` ES el arbol de archivos del paquete, asi que de ahi sale el `.deb` o el `.rpm` igual que desde el arbol de fuentes: se desempaqueta, se le pone el `DEBIAN/` o la cabecera de rpm, y se instala con el gestor de la maquina. Si no hay ni `dpkg` ni `rpm`, se instala el propio `.tgz` con `instalar-tgz`, que tambien viaja adentro. No se arma el `.tgz`, que ya esta, ni se compila mem_util, que sin fuentes no se puede.

Del tgz tambien se saca `usr/bin/instalar-tgz` si el paquete es anterior al 02/11/2026 y no trae `hacer-pkg-minimo`; y si no trae ninguno de los dos, se desempaqueta el tgz en la raiz, que es lo que hacia el Uinstall. Los tres caminos dicen por cual van.

El `.deb` y el `.rpm` que salen de ahi llevan **`~minimo` en la version**, aunque el tgz sea el estandar: traen los mismos archivos que el paquete normal pero no declaran dependencias --no pueden, porque existen para arrancar una maquina donde no se puede bajar nada--, asi que no son el paquete de la distribucion, y sin la marca el gestor los veria al dia y el completo no los

actualizaria nunca. El `.tgz`, en cambio, se instala con su version estandar: ese es el paquete de verdad.

El nombre del archivo dice el nombre, la version y la arquitectura (`NOMBRE_VERSION_ARQ.tgz`). Los datos del paquete --resumen, responsable, licencia, enlace-- salen del `summapack.summa`, que desde el 02/11/2026 viaja dentro del paquete, en `usr/share/summapack/data`: antes se quedaban en el arbol y sin arbol no habia de donde sacarlos.

Hacen falta `base64`, `tar`, `gzip` y `bash`; sin alguno de los cuatro el paquete no se puede sacar de adentro y lo dice.

G.5. Como se escribe cada formato a mano

G.5.1. El `.deb`

Un `.deb` es un archivo `ar` con tres miembros, en este orden: `debian-binary` (dice `2.0`), `control.tar.gz` y `data.tar.gz`. El `ar` se escribe en el propio guion, para no depender ni de `binutils`: son ocho bytes de magia (`!<arch>\n`) y despues, por miembro, una cabecera de 60 bytes de texto plano --nombre (16), fecha (12), uid (6), gid (6), modo (8), tamanio (10), fin (2)-- seguida del contenido, con un byte de relleno si el tamanio es impar, porque los miembros van en posiciones pares. Los dos `tar` son `tar` y `gzip`, que estan siempre.

Si `dpkg-deb` esta, se usa `dpkg-deb --root-owner-group --build`, que es el exacto; el `ar` escrito a mano es para cuando no esta. El formato 2.0 es el que `dpkg` lee desde 1999.

El `control` se escribe con los datos del `.summa`, y se le agregan los `md5sums`, el `copyright` que pide la politica de Debian, y los manuales, `info` y documentacion comprimidos como los comprime `debhelper` (los changelog siempre; el resto de `doc` si pasa de 4k; el `copyright` nunca).

G.5.2. El `.rpm`

Lo escribe `share/summapack/data/armar-rpm.sh`, sin `rpmbuild`:

```
armar-rpm.sh --raiz-archivos DIR --salida A.rpm --nombre N --version V
             --release R [--arch noarch] [--summary S] [--descripcion D]
             [--licencia L] [--grupo G] [--url U] [--empaquetador E]
             [--excluir D1,D2]
armar-rpm.sh --leer A.rpm       (muestra los metadatos con rpm -qip)
```

Un `.rpm` binario son cuatro partes: el *lead* de 96 bytes, la cabecera de firma, la cabecera con las etiquetas, y la carga, que es un `cpio` en formato `newc` comprimido con `gzip`. Cada cabecera es una lista de entradas de 16 bytes (etiqueta, tipo, desplazamiento, cantidad) y un area de datos; los `INT16`, `INT32` e `INT64` van alineados. Las dos cabeceras llevan su region --`HEADERSIGNATURES` (62) y `HEADERIMMUTABLE` (63)--, que es la primera entrada del indice y un remolque de 16 bytes al final del area de datos cuyo desplazamiento es `-(16 * cantidad de entradas)`. Asi las escribe `rpmbuild`.

No firma, porque no hay clave. Pone el tamanio, el tamanio de la carga sin comprimir y el md5 de cabecera mas carga, que es lo que `rpm` comprueba de un paquete sin firma. `SOURCERPM` va siempre: sin esa etiqueta `rpm` toma el paquete por un paquete de fuentes.

Cada gestor nombra distinto la misma arquitectura: `amd64` en dpkg, `x86_64` en rpm; `all` en dpkg, `noarch` en rpm. Y el nombre del archivo es `nombre-version-release.arq.rpm`, con guion antes del release, no punto.

`--leer` es un lector independiente, escrito para comprobar lo que se escribio: recalcula el md5 y compara la lista de archivos de la cabecera con la del `cpio`.

G.5.3. El `.tgz`

`tar` y `gzip`, con el nombre que espera `instalar-tgz`: `NOMBRE_VERSION_ARQ.tgz`. Lo instala `bin/instalar-tgz`, el instalador propio de summapack (el viejo «Uinstall» de 2011), que lleva

el registro en var/lib/summapack/tgz/ --NOMBRE.lista, NOMBRE.version, NOMBRE.respaldo-- y por eso puede quitar lo que puso.

G.6. mem_util compilada a mano

El paquete normal compila la biblioteca con autotools, porque es un subproyecto (mem_util/configure.ac). Aca no hay autotools, asi que se compila a mano con el compilador que haya ($CC, gcc o cc):

- las seis fuentes de mem_util/src: util, mem, time, datetime, abs2rel, rel2abs, con -O2 -fPIC;
- el .so con el soname que deja libtool con version-info 0: libmem_util.so.0.0.0, con -Wl,-soname,libmem_util.so.0, y los enlaces libmem_util.so.0 (en libmem-util0) y libmem_util.so (en el -dev);
- el .a con ar rcs;
- los encabezados util.h, mem.h, time_mem.h, pathconv.h;
- el mem_util.pc que normalmente genera configure, hecho con sed desde mem_util.pc.in, con la version que dice el AC_INIT del configure.ac.

Todo va al libdir que corresponde a la distribucion: el de gcc -print-multiarch si lo hay (/usr/lib/x86_64-linux-gnu), si no /usr/lib64 en los sistemas de 64 bits que lo tengan, y si no /usr/lib.

Si no hay compilador se avisa y se sigue, con los tres paquetes de summapack solo, que es lo que importa para arrancar; los proyectos que usen la biblioteca la van a pedir despues. No corta: pedido del autor.

G.7. Que se midio, y con que

Todo esto es del 02/11/2026, en Ubuntu con dpkg y sin rpm. El arbol de prueba es una granja de enlaces al arbol, para no dejar nada en el.

El .deb escrito a mano lo lee el dpkg de verdad. Armado con un PATH sin dpkg-deb:

```
$ file summapack_..._all.deb
Debian binary package (format 2.0), with control.tar.gz , data compression gz
$ dpkg-deb --info summapack_..._all.deb
new Debian package, version 2.0.
size 186200 bytes: control archive=2943 bytes.
    833 bytes,    17 lines      control
   5171 bytes,    77 lines      md5sums
$ dpkg -x summapack_..._all.deb x/          # extrae bien
```

Las 122 entradas (archivos, enlaces y directorios) que declara el escrito a mano son las mismas que las del armado con dpkg-deb, con los mismos permisos, tamanios y destinos de enlace; y el sp extraido es byte a byte el del arbol, y corre. Hay una diferencia: dpkg-deb guarda el dueno como root/root y el tar de aca lo guarda como 0/0 --mismo uid y gid 0, sin el nombre--, porque se usa --numeric-owner. No cambia lo que se instala.

El .rpm escrito a mano se lee entero y es consistente. Con su propio lector:

```
$ file summapack-....noarch.rpm
RPM v3.0 bin i386/x86_64
$ armar-rpm.sh --leer summapack-....noarch.rpm
Name        : summapack
Version     : 2026.2.1...
Signature   : (none)
$ rpm -K summapack-....noarch.rpm
summapack-....noarch.rpm: digests OK
```

Y la carga la extrae el `cpio` de GNU, con el `sp` extraido byte a byte igual al de origen.

El `.tgz` se instala con el registro. `instalar-tgz --raiz` con el de summapack y el de libmem-util0 deja las 105 y 5 entradas anotadas en `var/lib/summapack/tgz/`, y agrega el libdir a `ld.so.conf.d/summapack.conf`.

`--forma directo` pone los archivos. 119 archivos y enlaces en la raiz de prueba, los 117 del registro, con `usr/bin/sp` y `usr/lib/x86_64-linux-gnu/libmem_util.so.0.0.0` en su lugar.

La biblioteca compilada a mano anda. Queda igual que la del paquete normal --mismas rutas, soname `libmem_util.so.0`, `mem_util.pc` con el libdir de la distribucion, que `pkg-config` lee y da version 1.0-- y pasa su propia prueba: `t_datetime` de `mem_util/src`, compilado y enlazado contra ella, da

```
55 casos y 8000 conversiones concurrentes; 0 fallos
```

con codigo de salida 0.

El `sp` suelto instala sin arbol, sin red y sin gestor. Armado con `distros/armar-sp` (436 K, con el tgz de 250 K adentro), copiado a un directorio vacio donde no hay nada mas que el, y corrido con un PATH sin `dpkg` ni `rpm`:

```
$ PATH=.../sin-dpkg-ni-rpm sh sp --sin-red --raiz .
sp: instalando summapack_...~minimo_all.tgz, que viene adentro de este sp
hacer-pkg-minimo: ... formas deb rpm, desde /tmp/.../summapack_...tgz
hacer-pkg-minimo: sin dpkg-deb: el .deb se escribe a mano
hacer-pkg-minimo: armado /tmp/.../summapack_...~minimo_all.deb
armar-rpm: /tmp/.../summapack-...~minimo.noarch.rpm, 125 archivos
hacer-pkg-minimo: de los armados se instalan los tgz
hacer-pkg-minimo: instalando: .../usr/bin/instalar-tgz --raiz . ...
```

Termina con 0 y deja 111 archivos y enlaces, los 109 del registro, con el `sp` instalado byte a byte igual al del arbol y corriendo. El `bajar-paquetes` que saca de entre las dos marcas tambien es byte a byte el del arbol.

El deb rearmado desde un tgz es el mismo paquete. Rearmado desde el tgz estandar de r6055 y comparado con el `.deb` original de esa misma version, que armo `dpkg-buildpackage`: **las 120 entradas identicas**, mismos permisos, tamanios, nombres y destinos de enlace. En el control cambia lo que tiene que cambiar: no declara dependencias, la descripcion dice que es el paquete minimo y la version lleva `~minimo`. El `Installed-Size` se cuenta como lo cuenta dpkg-gencontrol --cada archivo redondeado al kB mas un kB por directorio--: da 643 donde el normal declara 647.

Sin compilador avisa y no corta. Con un PATH sin compiladores dice «sin compilador no se arman libmem-util0 ni libmem-util-dev; sigue summapack solo» y termina bien, con los tres paquetes de summapack.

G.8. Que NO se midio

- **La instalacion con `dpkg -i` o `rpm -U` no se corrio**: las dos piden root siempre, incluso con `--root` en un directorio propio, porque llevan su propia base de datos (`dpkg: error: requested operation requires superuser privilege`). Lo medido es que el paquete se arma y que `dpkg` lo lee entero. La instalacion sin root que si se midio es la del `.tgz` con `instalar-tgz` y la de `--forma directo`.
- **`rpm -i` del `.rpm` escrito a mano nunca se corrio**, porque en esta maquina no hay `rpm` y aca no se instala nada. Lo que se comprobo es la estructura, con `file` y con el lector propio. Falta probarlo en una maquina con `rpm`, que es donde se va a usar.
- La instalacion de verdad en `/` (sin `--raiz`) no se probo: todas las pruebas van a una raiz aparte.
- El recorrido completo del paso 1b se probo con PATH recortados, no en una maquina realmente sin red y sin herramientas.

G.9. Los archivos

```
sp/sp                                  la autoinstalacion (pasos 1, 1b, 2, 3)
bin/hacer-pkg-minimo                   los tres paquetes a mano
share/summapack/data/armar-rpm.sh      el .rpm a mano, y su lector
bin/instalar-tgz                       instala el .tgz, con registro
bin/sp-version                         la version del arbol
summapack.summa                        ARCHIVO_summapack_install y los datos
hacer-pkg                              el paquete normal, cuando si hay
                                       herramientas
bin/sp-bootstrap.in                    el arranque del sp suelto
distros/armar-sp                       lo arma, con el tgz adentro
```

H. Instalar paquetes de Simusol sin summapack

La forma de instalar es con summapack (`installsumma.sh` y `bajar-paquetes`; ver PAQUETES.md). Esto es para cuando no se puede o no se quiere: bajar a mano de `https://www.simusol.org/downloads/` e instalar. Pedido del autor, 07/10/2026: el bloque que estaba en la pagina de Tabular se saco de alli (se instala solo con summapack) y queda aca.

H.1. Como esta organizado el servidor

```
downloads/
  PROYECTO.summa      el .summa estable del proyecto: PACKAGES dice sus
                      paquetes y VERSION la version estable (6.0-38)
  PROYECTO/           los paquetes estables: PAQ_VERSION_ARQ.deb,
                      PAQ-VERSION.ARQ.rpm, PAQ_VERSION_ARQ.tgz,
                      PROYECTO_VERSION.tar.gz
    otros/            los .md5 de todos, lo inestable y lo anterior
  viejos/             lo que habia antes de octubre de 2026
```

ARQ es amd64 o all en los .deb y .tgz, x86_64 o noarch en los .rpm. Un paquete esta en el directorio del proyecto que lo arma: libmem-util0 en summapack/, libslex0 en slex/, libtabular2-0 en tabular2/.

H.2. Paso a paso

1. Ver la version estable del proyecto:

   ```
   curl -fsSL https://www.simusol.org/downloads/tabular2.summa | grep -E '^(PACKAGES
       |VERSION)='
   ```

2. Bajar cada paquete y su .md5, y comprobarlo:

   ```
   curl -fsSLO https://www.simusol.org/downloads/tabular2/tabular_2.0-14_amd64.deb
   curl -fsSLO https://www.simusol.org/downloads/tabular2/otros/tabular_2.0-14_amd64
       .deb.md5
   md5sum -c tabular_2.0-14_amd64.deb.md5
   ```

3. Instalar: `sudo apt-get install ./*.deb` (Debian/Ubuntu), `sudo dnf install ./*.rpm` (Fedora), o desempaquetar el .tgz en / (lo hace con registro, para poder quitarlo, `instalar-tgz` de summapack).

H.3. Ejemplo: Tabular en Ubuntu 24.04 amd64

Baja los cinco paquetes necesarios para ejecutar Tabular, los comprueba e instala tambien gnuplot. Las versiones son las de octubre de 2026 (ver el paso 1 para las actuales).

```
(
  set -euo pipefail
  . /etc/os-release
  if [ "${ID:-}" != ubuntu ] || [ "${VERSION_ID:-}" != 24.04 ] || [ "$(dpkg --print-
    architecture)" != amd64 ]; then
    printf '%s\n' 'Estos paquetes estan preparados para Ubuntu 24.04 amd64.'
    exit 1
  fi

  sudo apt-get update
  sudo apt-get --no-remove install ca-certificates curl

  d=$(mktemp -d /tmp/tabular-instalacion.XXXXXXXX)
  chmod 755 "$d"
  printf 'Descargas en: %s\n' "$d"
  paquetes=(
    summapack/libmem-util0_2017.2.1.R1008-7_amd64.deb
    slex/libslex0_1.0-24_amd64.deb
    unidades/libunidades0_1.0-1_amd64.deb
    tabular2/libtabular2-0_2.0-14_amd64.deb
    tabular2/tabular_2.0-14_amd64.deb
  )
  archivos=()
  for p in "${paquetes[@]}"; do
    proyecto=${p%%/*}; archivo=${p##*/}
    curl --fail --show-error --location --retry 3 --connect-timeout 15 --max-time 180
    \
      "https://www.simusol.org/downloads/$p" --output "$d/$archivo"
    curl --fail --silent --show-error --location \
      "https://www.simusol.org/downloads/$proyecto/otros/$archivo.md5" --output "$d/
    $archivo.md5"
    (cd "$d" && md5sum --check --quiet "$archivo.md5")
    archivos+=("$d/$archivo")
  done

  sudo apt-get --no-remove install gnuplot "${archivos[@]}"
  printf '%s\n' 'Tabular instalado. Ejecutar: interpola'
)
```

El bloque se detiene si falla una descarga, un md5 o la instalacion, y conserva los archivos en el directorio temporal que anuncia. No fuerza cambios de version ni elimina paquetes: revisar lo que propone APT antes de confirmar.

I. Inventario de proyectos con Summa

Este inventario distingue los proyectos actuales que tienen un `.summa` y `MANIFIESTO` de las copias históricas y los Makefiles propios. Se verificó contra los árboles locales de `siru` y `calcula` el 14 de noviembre de 2026. `PACKAGES` enumera los paquetes binarios declarados; un proyecto puede producir varios. Radia se incorporó después de la verificación inicial de los 13 proyectos.

Proyecto	Paquetes declarados en `PACKAGES`	Árbol	Archivos registrados en prueba local
dia-simusol	dia-simusol	`siru/dia-simusol`	19
exprtk	libexprtk-dev	`siru/exprtk`	3
lanzador	lanzador	`siru/lanzador`	24
sceptresimusol	sceptresimusol	`siru/sceptresimusol`	123
simplot	simplot	`siru/simplot`	191
simusol	simusol	`siru/simusol`	375
siru	siru	`siru/siru`	416
slex	libslex0, libslex-dev, d2datdinc	`siru/slex`	25
summapack	summapack, libmem-util0, libmem-util-dev	`siru/summapack`	125
tabular2	libtabular2-0, libtabular2-dev, tabular	`siru/tabular2`	29
unidades	libunidades0, libunidades-dev	`siru/unidades`	17
calcula	calcula	`calcula/calcula-6.0`	60
puyuspa	puyuspa	`calcula/puyuspa-7.0`	72
Radia	radia	`siru/Radia`	33

La cadena que construye `summapack/distros/armar-paquetes` comprende `summapack`, `unidades`, `exprtk`, `slex`, `tabular2`, `simplot`, `calcula`, `siru` y `puyuspa`. Los otros cuatro proyectos de la tabla también tienen `.summa` y `MANIFIESTO`, pero no entran en esa cadena predeterminada. Hay además versiones antiguas y proyectos con `.summa` y Makefile propio fuera de esta tabla; no se consideran parte de la cadena actual.

I.1. Verificación del 14 de noviembre de 2026

Prueba	Resultado
`make install` directo en copias limpias de los 13 proyectos, con prefijos temporales en Ubuntu 24.04	Los 13 crearon registro y desinstalador ejecutable; las rutas registradas existían.
Ejecutar cada desinstalador	Los 13 eliminaron los archivos instalados, su registro y su propio script; quedaron cero archivos o enlaces en cada prefijo temporal.
Generar `debian/` en los 13 proyectos	Correcto; se armó además el DEB de SummaPack en Ubuntu.
Generar y expandir `.spec` para Fedora y SUSE en los 13 proyectos	Correcto con `rpmspec`; se armaron RPM de SummaPack con ambas marcas mediante `rpmbuild` en la máquina Ubuntu, como prueba del generador.
Construir TGZ de SummaPack	Correcto en la máquina Ubuntu.
Probar dentro de las imágenes Ubuntu, Trisquel, Fedora, SUSE y Arch	Correcto: `sp/tests/test_generadores_podman.sh ubuntu trisquel fedora suse pacman` pasó instalación, desinstalación y paquete nativo en las cinco imágenes.

Prueba	Resultado
Probar dentro de LFS	Pendiente: la construcción de la raíz se detuvo en `702-kernfs` al montar sistemas virtuales dentro de Podman; no hay imagen LFS validada.
Radia en Trisquel	Se construyó y probó el DEB: instalación, carga del módulo Perl y retirada.
Radia en Fedora y SUSE	Se construyeron RPM nativos, se instalaron, cargó el módulo Perl y se retiraron.

Para repetir las pruebas dentro de los contenedores, ejecutar `summapack/sp/tests/test_generadores_podm`
Se pueden indicar formas concretas, por ejemplo `fedora suse trisquel`.

J. Evaluación de migración de comandos a Bash

SummaPack usa principalmente Bash o POSIX sh. Los comandos no Bash y los motores que invocan los comandos Bash son los siguientes. Las cifras son líneas actuales del archivo, no una estimación de trabajo. Esta evaluación no autoriza sustituirlos sin pruebas de equivalencia.

Programa	Lenguaje y tamaño	Función y dificultad de migración
`bin/templatesumma.pl`	Perl, 126 líneas	Baja en código, pero usa Template Toolkit y pertenece a las reglas históricas de `makefile.lib`. Las operaciones actuales del `.summa` usan `summa.sh`; queda por retirar este generador heredado.
`viejo/bin/packinstall.pl`	Perl, 775 líneas	Migrado en el uso actual: `bin/packinstall.sh` es un adaptador Bash de `sp`. El original se conserva como antecedente; sus familias sin packlet no se ejecutan silenciosamente.
`bin/manifiesto-instalacion`	Python, 163 líneas	Media-alta. Registra rutas y huellas SHA256 en JSON y retira solo archivos intactos; migrarlo a Bash requiere `jq`, `sha256sum` y especial cuidado con nombres de archivo y borrados. Lo usan `make install/uninstall`.

Programa	Lenguaje y tamaño	Función y dificultad de migración
`share/summapack/data/summa.sh`	Bash	Migrado. Coordina consultas, creación, actualización y verificación del `.summa`, además de los generadores Bash para DEB, RPM y Automake. El programa Python anterior está archivado en `viejo/summa.py`.
`share/summapack/data/sincronizar_web.py`	Python, 114 líneas	Media por sí solo, pero depende del motor siguiente y de configuración JSON.
`share/summapack/data/subir_pagina.py`	Python, 543 líneas	Muy alta. Sincroniza tres estados MD/web/base, normaliza HTML, conserva respaldos y usa SSH; Bash necesitaría `jq`, `pandoc` y un analizador HTML externo sin perder la detección de conflictos.
`doc/web/generar.py`	Python, 236 líneas	Alta. Genera la página TinyMCE, procesa HTML y llama a Pandoc; es herramienta de documentación, no comando instalado de usuario.
`doc/web/verificar.py`	Python, 89 líneas	Media. Analiza HTML, enlaces y huellas; depende de `generar.py`.

Hay además programas C de `mem_util`: no son comandos de administración y no corresponde traducir esa biblioteca a Bash.

La sustitución de `packinstall` ya se hizo: acepta la sintaxis histórica y delega en `sp` las instalaciones de paquetes del sistema, módulos Perl, repositorios y proyectos Simusol conocidos. `summa.sh` reemplaza el uso activo de `summa.py`; se comparó la salida de `crear` en ocho paquetes y `probar` devolvió `igual` en esos ocho. Quedan otros programas Perl y Python, especialmente los de sincronización web.

K. Sincronizar Markdown y CMSimple XH

`sincronizar-pagina SISTEMA [--prueba]` es la entrada única. Solo recibe el nombre del sistema y, opcionalmente, `--prueba`: todos los demás datos están en una tabla central. Compara el Markdown del proyecto y la página editada en la web. **Antes de regenerar cualquier HTML consulta la web.** No decide por fechas: guarda la última versión común.

El HTML del árbol no se edita a mano: se genera desde MD o se baja de la web. La herramienta también conserva un modo de publicación unidireccional sin `--sincronizar`, para fragmentos preparados explícitamente; ese modo no decide qué fuente cambió.

K.1. Herramientas, nombres y lenguajes

Las herramientas propias pertenecen a SummaPack. El usuario trabaja con un solo comando; el motor compartido hace las conversiones, subidas y bajadas.

Programa	Función	Estado actual	Implementación
`sincronizar-pagina SISTEMA`	Consultar la web, comparar MD/web y coordinar todas las páginas configuradas del sistema.	Implementado. Solo admite el sistema y `--prueba`, además de la ayuda.	Lanzador Bash, tabla JSON y motor Python 3; transporte SSH.
`subir-pagina`	Publicación de HTML y acceso a opciones avanzadas del motor.	Se conserva por compatibilidad; no hace falta usarlo en el trabajo habitual.	Comparte el motor Python 3.
Bajada de páginas	Página CMSimple → HTML local y MD.	Operación interna de `sincronizar-pagina`; no requiere un comando `bajar-pagina` separado.	Mismo motor Python 3.
`pandoc`	MD → HTML y HTML descargado → MD.	Dependencia externa utilizada por el motor; no es un programa propio de SummaPack.	Pandoc está escrito en Haskell; SummaPack lo invoca como ejecutable.

Bash lanza el motor; Python 3 lee la tabla JSON, compara versiones, usa SSH, respalda y actualiza archivos. No hace falta programar en Haskell para mantener SummaPack.

K.2. Decisión automática

Desde la última sincronización	Acción
No cambió MD ni web	No publica ni convierte; recupera el HTML de la web si falta o difiere.
Cambió solo MD	Consulta primero la web, convierte MD a HTML con Pandoc y publica.
Cambió solo web	No regenera desde el MD viejo. Descarga HTML, lo convierte a MD y conserva ambos.
Cambiaron MD y web	Se detiene sin convertir ni escribir. Muestra las diferencias contra la base.
No hay base común	Usa la política `inicio` de la tabla: `revisar` se detiene; `web` importa; `local` requiere una huella web revisada.
No se puede consultar la web	Falla sin regenerar ni sobrescribir archivos.

Las conversiones usan Markdown GFM. Al bajar, el MD incorpora el contenido web para que una regeneración futura no elimine las ediciones de TinyMCE. Pandoc puede cambiar presentación, estilos, atributos o construcciones que no tengan equivalente en Markdown: no se promete una ida y vuelta idéntica de HTML. Revisar la vista previa, especialmente tablas,

código y elementos especiales. El respaldo conserva siempre el HTML descargado original y el MD anterior.

K.3. Orden completo de una sincronización

1. **Consultar la página web antes de convertir o regenerar HTML.** Si no se puede leer el servidor, detenerse sin reemplazar los archivos locales. Leer no publica ni modifica el CMS.
2. Leer el MD local y el registro `base` configurado para esa página. Comparar el contenido actual de cada lado con su propia versión guardada, no sus fechas.
3. Aplicar la tabla de decisión. Si cambiaron ambos lados, detenerse antes de convertir. Si falta base, aplicar la elección inicial declarada en la tabla; sin elección, detenerse.
4. Si cambió solo MD, convertirlo a HTML en memoria. Si cambió solo la web, conservar el HTML descargado y preparar su conversión a MD, sin regenerar desde el MD viejo. Comprobar que el resultado sea un fragmento admisible.
5. Antes de reemplazar archivos, conservar los respaldos. Al publicar, respaldar también el contenido completo del CMS fuera de la raíz pública.
6. Aplicar en la dirección elegida. Antes de escribir en la web, comprobar bajo bloqueo que su huella SHA256 siga siendo la consultada; si cambió entretanto, rechazar la publicación. Antes de importar, comprobar que el MD local no haya cambiado durante la comparación.
7. Guardar la nueva base local **solo después de completar la actualización del contenido y de sus archivos locales**. Si una operación falla, no declarar que la sincronización terminó: conservar respaldos, revisar el estado parcial y volver a comparar.

`--prueba` consulta y prepara diferencias sin reemplazar archivos ni crear una base. Se revisan todas las páginas configuradas antes de aplicar alguna; si alguna tiene conflicto, no se aplica el conjunto. Al aplicar se vuelve a consultar cada página. No hay una transacción global: si ocurre un fallo después de actualizar una página, las anteriores pueden haber quedado actualizadas. No hay proceso en segundo plano.

K.3.1. Qué significa la base común

Registro	Qué conserva	Para qué sirve
Base local indicada por `base` en la tabla	Identidad del servidor, usuario, raíz, página, rutas locales y las versiones MD y HTML web de la última sincronización completada.	Saber qué lado cambió, sin comparar directamente Markdown con HTML.
Estado remoto de publicación	Huella de la página publicada y referencia a su respaldo.	Detectar ediciones posteriores y proteger una nueva escritura sobre CMSimple.
HTML del árbol indicado por `archivo` en la tabla	Resultado generado desde MD o descargado de la web.	Usarlo como salida; nunca decide por sí solo la dirección de sincronización.

La base conserva **un par correspondiente de versiones**, no exige que MD y HTML sean iguales ni que una conversión de ida y vuelta conserve exactamente el formato de TinyMCE. Tras una descarga se registran el MD importado y el HTML recibido; por eso la siguiente corrida sin cambios no vuelve a convertir ni a publicar por diferencias de formato.

Si no cambió ninguna fuente, no se actualiza el sitio ni el MD. Puede recuperarse el HTML local desde la versión web consultada si falta o difiere. Si falla un paso después de escribir uno

de los lados, no hay una transacción distribuida que deshaga automáticamente todo: la base anterior y los respaldos permiten reconocer y revisar la situación.

K.3.2. Implementado y pendiente

- Implementado: comando único `sincronizar-pagina SISTEMA`, tabla central y decisión MD/web, consulta previa, conversiones, respaldos, conflictos y modo de prueba.
- Pendiente: completar las entradas de Calcula y Puyuspa y las subpáginas adicionales. No se adivinan destinos ni se crean páginas.
- Pendiente: conectar o migrar los generadores antiguos de cada proyecto. Mientras tanto, ejecutar un `generar.py` por separado no consulta automáticamente la web y puede invertir el orden requerido.
- Fuera de este flujo: crear páginas, cambiar su jerarquía o menú, fusionar automáticamente cambios simultáneos, programar sincronizaciones periódicas o reemplazar un sitio completo.

K.4. Requisitos y configuración

- Cliente: Bash, Python 3, OpenSSH y Pandoc (este último solo para sincronizar MD).
- Servidor: Python 3, acceso SSH y permisos de lectura/escritura sobre el contenido.
- CMS: **CMSimple XH 1.8** con metadatos embebidos en `content/content.htm`. Otras versiones o formatos se rechazan hasta verificarlos.
- La página ya debe existir. No crea, borra, renombra ni mueve páginas; no cambia el menú.

K.5. Tabla central

La tabla distribuida está en `share/summapack/data/paginas-web.json` (instalada: `/usr/share/summapack/data/paginas-web.json`). Puede sustituirse completa por `~/.config/simusol/paginas-web.json`. Para pruebas aisladas, `SUMMAPACK_PAGINAS` selecciona otra tabla; no es un argumento habitual.

Campo	Contenido
`version`	Versión del esquema, actualmente 1.
`raiz_local`	Raíz de los árboles locales, por omisión `~/svn`. Las rutas relativas de archivos se resuelven desde ella.
`conexion`	`servidor`, `usuario` SSH, `raiz` del CMS y opcionalmente `estado` remoto. No contiene contraseñas.
`sistemas`	Una entrada por nombre: `slex`, `tabular2`, `summapack`, `calcula`, `puyuspa`, etc.
`paginas`	Lista de páginas del sistema, cada una con `pagina`, `md`, `archivo` y `base`; puede tener su propia `conexion`.
`inicio`	Solo sin base: `revisar` se detiene; `web` importa la web, creando MD si falta; `local` publica MD y exige `aceptar_actual` con el SHA256 revisado. Con base existente se decide automáticamente.
`pendiente`	Motivo por el que el sistema aún no se puede sincronizar. Se informa antes de conectar.

El estado remoto queda por omisión en `~/.local/state/summapack/cmsimple` de la cuenta SSH: fuera de la raíz web, propio y con modo 700. El registro `base` es local y distinto del estado remoto. El comando nuevo toma los datos de esta tabla, no del archivo de publicación de paquetes `~/.config/simusol/subir`.

Configuración inicial incluida:

Sistema	Páginas / situación	Inicio
`slex`	Página existente `Slex`; crea `siru/slex/doc/web/pagina.md` desde la web.	web
`summapack`	Solo portada existente `Summapack`; crea `siru/summapack/doc/web/pagina.md`. Las subpáginas no están incluidas aún.	web
`tabular2`	Página `TABULAR` y MD existente; no sobrescribe ninguna versión inicial sin revisar.	revisar
`calcula`	Falta definir fuente y destino propio; existe una página compartida Psicro-Calcula.	pendiente
`puyuspa`	Falta definir fuente y página CMSimple; no confundir con la aplicación Puyuspa.	pendiente

K.6. Uso: un solo comando

```
sincronizar-pagina slex --prueba
sincronizar-pagina slex
```

Sin instalar todavía, desde cualquier directorio: `~/svn/siru/summapack/sincronizar-pagina slex --prueba`. Se reemplaza `slex` por el sistema configurado. No se pasan rutas, preferencias ni destinos como parámetros.

`--prueba` consulta y muestra diferencias sin escribir. Revisar y repetir sin esa opción para aplicar. Si falta base, revisar la política `inicio` de la tabla; no modificarla para ocultar un conflicto con una base existente. Un conflicto se resuelve revisando las dos versiones: no se fusiona HTML automáticamente.

Conservar el MD y la base por página. La identidad incluye las rutas absolutas y el usuario: al trasladar el trabajo a otra ubicación hay que revisar esa identidad, no copiar ciegamente la base. No ejecutar dos sincronizaciones simultáneas sobre el mismo par MD/base. No ejecutar antes el generador antiguo. El servidor no necesita instalar SummaPack: el worker viaja por SSH y no instala un endpoint web.

K.7. Respaldos y garantías

Recurso	Conservación
Sitio remoto	Antes de publicar guarda `content.htm` completo y los fragmentos anterior/posterior en un directorio privado fuera de la web.
Fuente local	Antes de aplicar guarda MD, HTML descargado, salida anterior si existe y base anterior en `respaldo-cmsimple-*` junto a la base.
Metadatos de la página	Conserva URL, menú, nivel, publicación y demás opciones; solo actualiza `last_edit` al publicar.
Otras páginas	Sus bytes se conservan; no se reserializa el sitio entero con un parser HTML.
Escritura remota	Usa `flock` sobre el mismo archivo que CMSimple y verifica el SHA256 antes de escribir. Conserva inode, propietario y permisos.

No abrir el editor de esa página mientras se sincroniza: un formulario que quedó abierto antes podría guardarse después y reemplazar la publicación. El bloqueo evita escrituras simultáneas, pero no impide que una sesión del CMS guarde luego un texto viejo. La siguiente sincronización lo detectará como cambio web.

La escritura de `content.htm` se hace sobre su inode original, como CMSimple, para compartir sus bloqueos. No es una transacción resistente a cualquier caída de energía: existe un respaldo previo y se intenta restaurar ante errores recuperables. Una falla entre la publicación, la escritura local y la actualización de la base puede exigir revisión manual; no reintentar forzando sin mirar las versiones. La herramienta no ejecuta el PHP del contenido.

Para recuperar, usar los archivos de respaldo después de comparar con el estado actual. **No reemplazar ciegamente todo `content.htm` por un respaldo antiguo**, porque perdería cambios posteriores de otras páginas. La herramienta imprime la ruta del respaldo y no los borra automáticamente.

K.8. Generadores existentes y alcance

Este flujo trabaja con **un MD completo por página**, incluyendo su primer título en el nivel correspondiente. No interpreta plantillas, inclusiones de C ni LaTeX. Los generadores anteriores de Slex, Tabular o SummaPack no se interceptan automáticamente: hay que migrar cada página a este flujo o conectar explícitamente su generador después de la consulta web. No ejecutar el generador antiguo y después sincronizar, porque invertiría el orden necesario.

Una portada o manual que todavía nazca de `.tex`, `inicio.html` y otros archivos requiere definir su fuente antes de migrarlo. No se convierte automáticamente un manual LaTeX en una fuente Markdown independiente ni se cambian archivos de otra sesión. El modo de publicación simple sirve para esos casos, pero no reemplaza la sincronización MD/web.

No se ha publicado ninguna página del servidor durante el desarrollo. Se verificó por SSH que el sitio real usa CMSimple XH 1.8 y que existe la página `Slex`; el listado y la comparación real fueron de solo lectura.

K.9. Pruebas y salidas

```
python3 pruebas/prueba-subir-pagina.py
bash -n subir-pagina
```

```
bash -n sincronizar-pagina
```

Las pruebas crean sitios ficticios en directorios temporales: publicación, respaldo, estructura, primera base, cambios MD/web, conflicto, orden de consulta, caída de SSH simulada, carrera antes de publicar y conversión de tablas/código con Pandoc. Nunca conectan al servidor. Devuelve 0 al completar o previsualizar una operación válida, 3 ante falta de base/conflicto, 2 para argumentos inválidos y 1 ante errores.

L. SummaPack

El proyecto SummaPack tiene todavía un grado de avance pequeño: apenas lo necesario para instalar los paquetes del proyecto Simusol.

SummaPack es un conjunto de herramientas para compilar, empaquetar, publicar e instalar software libre. Su idea central es mantener la información del empaquetado en el proyecto original y generar desde allí los formatos que necesitan diferentes sistemas.

El proyecto fue desarrollado por Diego Saravia y está vinculado a UTUTO, Simusol y Siru. Su nombre aparece en los documentos tanto como Sumapack como SummaPack.

Descargas e instalación La guía única de instalación (`https://www.simusol.org/downloads/0-README-como-instalar`) mantiene las instrucciones vigentes para todos los programas del proyecto. Para instalar Calcula directamente en Debian o Ubuntu:

```
curl -fsSLO https://www.simusol.org/downloads/sp
sh sp --para deb calcula
```

Una sola orden instala SummaPack, Calcula y sus dependencias; no hace falta instalar SummaPack antes. Sin nombre de proyecto se instala sólo SummaPack. Las opciones siguientes son alternativas:

Opción de `sp`	Recorrido
`--para fuentes`	Preferida: compila en la máquina e instala con `make install` en `/usr/local`, o en `--prefijo`, con registro para desinstalar.
`--para fuentes-paquete`	Compila y construye paquetes `.deb` o `.rpm` locales, instalados por el gestor de la distribución.
`--para deb`	Usa paquetes binarios para Debian y Ubuntu. Los demás formatos se describen en la guía.
`--prueba`	Permite revisar el recorrido sin instalar.

Después se usa `sp install calcula`, `sp remove calcula` o `sp search gsl`. La instalación y sus requisitos se mantienen en la guía enlazada; los binarios deben ser compatibles con la arquitectura y las bibliotecas del destino. `packinstall` y `bajar-paquetes` son componentes internos o de compatibilidad, no pasos adicionales para el usuario.

El proyecto actual En la reorganización de octubre de 2026, cada proyecto tiene un archivo `.summa` que reúne los metadatos, las dependencias y las instrucciones de empaquetado. La herramienta `hacer-pkg` construye paquetes DEB, RPM y TGZ y archivos de fuentes `.tar.gz`. La generación de RPM requiere `rpmbuild`; si no está disponible, se avisa y se omite ese formato. El manual describe el estado de esa implementación, sus límites y lo que todavía falta.

Herramienta	Función
`.summa` y `summa.py`	Describir el proyecto y generar la información de los paquetes.
`hacer-pkg` y `hacer-pkg.inc`	Construir fuentes `.tar.gz` y paquetes `.deb`, `.tgz` y, si está disponible `rpmbuild`, `.rpm`.
`subir-paquetes`	Publicar archivos por proyecto y administrar versiones estables e inestables.
`subir-doc`	Publicar los dos PDF de SummaPack en `downloads/summapack/docs/` del sitio de Simusol. Crea `docs/` solamente si el directorio del proyecto ya existe y respalda los PDF que reemplaza. No modifica las páginas de CMSimple.
`subir-pagina --sincronizar`	Sincronizar Markdown con una página existente de CMSimple XH 1.8. Consulta la web antes de regenerar HTML. Si cambió solo el MD, genera y sube; si cambió solo la web, baja e incorpora al MD; si cambiaron ambos, se detiene. Conserva una base común y respaldos. Requiere Python 3, SSH y Pandoc; guía completa en `doc/SINCRONIZAR-WEB.md` de las fuentes.
`instalar-tgz`	Instalar TGZ con registro de archivos, consultar lo instalado, actualizar y desinstalar con `--quitar`. Respalda los archivos que reemplaza.
`sp` (arranque público)	Arranque público autónomo: instalar SummaPack y los proyectos solicitados con sus dependencias mediante una sola orden.
`distro` (fuente: `distro.sh`)	Identificar el sistema con `uname -s` y leer `ID`/`ID_LIKE` de `/etc/os-release` sin ejecutarlo. Si falta, consulta `/usr/lib/os-release`; conserva detección histórica. `distro +` añade nombre y versión; `DISTRO_ROOT` permite diagnóstico sobre otra raíz. WSL informa su distribución real. El descargador no elige automáticamente el formato.
`packinstall` y `diccionariodepaquetes`	`packinstall` conserva la sintaxis antigua y delega en `sp`; el diccionario traduce nombres de dependencias entre distribuciones. `packinstall --prueba` muestra las órdenes sin ejecutarlas.
`sp`	Interfaz común para operaciones de paquetes. Detecta la distribución, traduce nombres con `diccionariodepaquetes` y delega en packlets APT, DNF, Zypper, Pacman, TGZ o SummaPack. Incluye `--ensayo` para revisar comandos sin ejecutarlos; los detalles están en la sección siguiente.
`makefile.lib`	Compartir reglas de construcción, instalación y preparación de dependencias entre proyectos.
`preparar`	Script que acompaña las fuentes: instala lo necesario para compilarlas mediante las herramientas internas de SummaPack. No construye el proyecto principal.
`fuentesexternas`	Catálogo de URL, versiones, SHA256, dependencias y recetas para construir bibliotecas y herramientas externas. Las entradas de base deben estar previamente instaladas; no se construye todo el sistema desde cero.
`distros/compilar-dentro.sh` y Containerfile	Construir paquetes en entornos Podman de Fedora, openSUSE y Arch; dejar RPM/TGZ y logs en `distros/salida/`. No publica automáticamente.
`summapack`	Consultar la versión instalada, la ubicación de sus datos y la ayuda de las herramientas.
`findbinlib`	Buscar programas, bibliotecas y módulos instalados manualmente que podrían ocultar los de un paquete nuevo.
`icon_install`	Instalar lanzadores de escritorio y entradas de menú mediante las herramientas XDG; admite `--prueba`.

Herramienta	Función
`templatesumma`	Herramienta heredada para generar `.changes` y `.spec` desde plantillas. Se conserva para las reglas antiguas; el flujo actual utiliza `summa.py` y `hacer-pkg.inc`.
`mem_util`	Biblioteca C para memoria, archivos, rutas, cadenas y fechas; se usa mediante `pkg-config --cflags --libs mem_util`. Fechas completas a segundos epoch, con zona local o explícita; horas sin fecha a segundos desde medianoche. Paquetes `libmem-util0` y `libmem-util-dev`.
`doc/web/generar.py`, `verificar.py` y `pruebas/`	Generar HTML para TinyMCE y vistas previas, verificar enlaces y PDF y probar las herramientas con casos controlados. No instalan ni publican por el hecho de generar la documentación.
`etc/`, `viejo/` y licencias	Configuración distribuida, programas retirados y documentos de autoría/licencia. No sustituyen la configuración personal de publicación; los programas retirados no son la interfaz recomendada.

Comandos del árbol de SummaPack La tabla distingue los comandos de uso general de los auxiliares de construcción: estos últimos se ejecutan desde el árbol de fuentes y no necesariamente se instalan en el `PATH`. Las órdenes de `make` no son programas independientes.

Comando	Qué hace
`summapack`	Muestra versión, rutas y ayuda del paquete instalado.
`distro`	Identifica la distribución y, con `+`, su versión.
`sp`	Administra paquetes mediante APT, DNF, Zypper o Pacman; `sp podman ENTORNO` prepara una de las diez imágenes base.
`sp` (arranque público)	Arranca la instalación de SummaPack y de los proyectos pedidos, en forma binaria o desde fuentes.
`bajar-paquetes`	Descarga paquetes y fuentes publicados, verifica sus sumas y puede instalarlos.
`instalar-tgz`	Instala, consulta, actualiza y desinstala paquetes TGZ con registro.
`packinstall`	Adapta la sintaxis antigua a `sp` para dependencias externas, módulos Perl, repositorios y proyectos Simusol; admite `--prueba`.
`instalar-del-summa`	Instala o desinstala en un destino los archivos declarados en un `.summa`, sin herramientas Debian.
`preparar`	Instala dependencias para construir el proyecto fuente que lo acompaña.
`summa.py`	Lee metadatos `.summa` y genera información para paquetes.
`hacer-pkg`	Construye archivos de fuentes y paquetes DEB, RPM y TGZ.
`crear-fuentes`	Genera un archivo de fuentes reproducible sin compilar.
`actualizar-paquetes`	Recorre proyectos, reconstruye solo los cambiados y publica y estabiliza sus paquetes.

Comando	Qué hace
`subir-paquetes`	Publica paquetes y cambia su estado entre inestable y estable.
`subir-doc`	Publica los PDF del proyecto con respaldo.
`sincronizar-pagina`	Compara Markdown y CMSimple; sube o baja el lado cambiado y se detiene ante conflictos.
`subir-pagina`	Publicación web directa con opciones avanzadas; se conserva por compatibilidad.
`icon_install`	Instala lanzadores de escritorio y entradas de menú XDG.
`findbinlib`	Detecta programas y bibliotecas instalados manualmente que ocultarían los del paquete.
`templatesumma`	Genera archivos de empaquetado con plantillas antiguas; no es el flujo actual.
`manifiesto-instalacion`	Registra y revierte una instalación manual de SummaPack sin borrar archivos modificados.
`publicar-cadena`	Auxiliar para construir, probar y publicar la cadena de paquetes de Simplot.
`distros/preparar-diez-entornos`	Invoca diez veces `sp podman`, una por imagen base vieja o actual.
`distros/exportar-diez-entornos`	Exporta las diez raíces limpias a archivos `tar.gz`.
`distros/comprobar-repos-desarrollo`	Prueba en un contenedor desechable que `sp buildtools` pueda instalar las dependencias.
`distros/preparar-desarrollo`	Crea copias de las imágenes base con compiladores, GSL, OpenBLAS y SuiteSparse.
`distros/compilar`	Construye paquetes de un proyecto dentro del contenedor de una distribución.
`distros/armar-bibliotecas`	Construye el paquete de bibliotecas externas comunes en el entorno de Ubuntu 22.04.
`distros/espejo-fuentes`	Descarga y verifica las fuentes externas para conservar un espejo.
`distros/dentro.sh`	Auxiliar interno invocado por `distros/compilar` dentro del contenedor.
`distros/bibliotecas.sh`	Auxiliar interno invocado por `distros/armar-bibliotecas`.

sp: interfaz común para operaciones de paquetes `sp podman ENTORNO` prepara una imagen base limpia de Podman. `ENTORNO` es `ubuntu`, `debian`, `fedora`, `suse` o `arch`, seguido de `-viejo` o `-actual`; por ejemplo, `sp podman suse-viejo`. Las diez referencias a imágenes están en el packlet de `sp`, no en el script que las recorre. `sp --ensayo podman suse-viejo` muestra las órdenes sin descargar. Desde el árbol de fuentes, `distros/preparar-diez-entornos` invoca esa orden para las diez variantes; `distros/preparar-desarrollo` crea aparte las imágenes con compiladores y bibliotecas, sin alterar las bases limpias.

`sp` es el programa de SummaPack para instalar, quitar, buscar e inspeccionar paquetes con una sintaxis común. Detecta la distribución mediante `/etc/os-release`, traduce nombres de referencia del diccionario de SummaPack y llama al gestor nativo mediante un packlet. Se instala como parte de SummaPack; su ayuda está disponible con `sp --help`, el manual Unix con `man sp` y el manual Texinfo con `info --file=/usr/share/info/sp.info`.

La forma general es `sp [--distro DISTRO] [--ensayo] ACCION [OBJETO] [ARGUMENTOS...]`. Las opciones globales deben preceder a la acción. `--distro` selecciona una distribución o una familia (`debian`, `fedora`, `opensuse` o `arch`); si se omite, se autodetecta el sistema. `--ensayo` muestra las órdenes que se usarían sin ejecutar el gestor ni cambiar paquetes. Las consultas no requieren privilegios; las operaciones de instalación o modificación utilizan `sudo` cuando hace falta.

Acción	Uso y resultado
`install PAQUETE...`	Instala paquetes por nombre. También acepta los objetos y conjuntos de mantenimiento de las tablas siguientes.
`remove PAQUETE...`	Quita paquetes instalados por su nombre. Admite además `REPO` y `TGZ`.
`clean WORLD` / `clean CACHE`	Elimina dependencias huérfanas o limpia la caché; alguna combinación puede no estar disponible en todos los gestores.
`fix WORLD o SYSTEM`	Ejecuta la reparación o sincronización equivalente que ofrece la familia.
`search [NAMES] TEXTO`	Busca paquetes; `NAMES` es opcional y `search all TEXTO` pide una búsqueda amplia cuando el gestor la implementa.
`info PAQUETE...`	Muestra datos de los paquetes. `info WORLD`, `SYSTEM`, `ALL` o `CACHE` consulta estadísticas/listados del gestor.
`buildtools`	Instala el conjunto de herramientas de desarrollo definido para esa familia.
`podman ENTORNO`	Descarga, etiqueta y comprueba la imagen base limpia de una de las diez variantes de distribución y época.
`update`, `distupdate`, `refresh`, `depclean`	Órdenes directas de los packlets para actualizar, sincronizar, refrescar índices o retirar dependencias huérfanas.
`addrepo REPO` / `delrepo REPO`	Añade o quita un repositorio en APT, DNF o Zypper. Pacman no implementa estas acciones.

Como objetos de `install`, `WORLD` actualiza paquetes, `SYSTEM` sincroniza la distribución y `CACHE` actualiza los índices del gestor. `install ALL` está reservado y todavía no tiene implementación.

Objeto	Ejemplo	Función
`PAQ` o `PACKAGES`	`sp install libgsl-dev`	Instala nombres lógicos del diccionario. El marcador puede omitirse.
`DEB`	`sp install DEB ./paquete.deb`	Instala un archivo DEB local en distribuciones APT.
`RPM`	`sp install RPM ./paquete.rpm`	Instala un RPM local en Fedora/DNF u openSUSE/Zypper.

Objeto	Ejemplo	Función
`TGZ`	`sp install TGZ ./paquete.tgz`	Instala mediante `instalar-tgz` en todas las familias soportadas; `sp remove TGZ NOMBRE` lo desinstala usando su registro.
`SUMMA`	`sp install SUMMA dibuja`	Instala un paquete SummaPack mediante `bajar-paquetes`.
`SOURCE`	`sp install SOURCE siru`	Obtiene e instala fuentes mediante `bajar-paquetes`.
`REPO`	`sp install REPO REPOSITORIO`	Administra repositorios en APT, DNF y Zypper; no está disponible en Pacman.

Familia de paquetes	Distribuciones reconocidas	Gestor
APT	Debian, Ubuntu y derivadas reconocidas como Linux Mint, Pop!_OS, Raspbian, Kali, elementary OS y Zorin	`apt-get`
DNF	Fedora, RHEL, Red Hat, CentOS, Rocky Linux, AlmaLinux, Oracle Linux y Nobara	`dnf`
Zypper	SUSE, openSUSE Leap/Tumbleweed, SLES y SLED	`zypper`
Pacman	Arch Linux, Manjaro y EndeavourOS	`pacman`

El diccionario usa nombres Debian como claves. Ubuntu y algunas derivadas consultan primero su columna `UBUNTU` y recurren a `DEBIAN` cuando no hay entrada propia; Fedora/DNF, SUSE/Zypper y Arch/Pacman consultan sus columnas respectivas. Una equivalencia `-` indica que la dependencia no hace falta en esa distribución: `sp` la omite al instalar dependencias y la informa como error si se solicita en otra acción. Los argumentos se pasan al gestor como argumentos separados, no como texto para ejecutar en un shell.

`sp --distro fedora --ensayo install libgsl-dev` permite comprobar la selección de familia y la traducción sin ejecutar DNF; `sp --ensayo buildtools` muestra el conjunto de desarrollo correspondiente a la máquina actual. En Arch, una instalación desde repositorios usa `pacman -Syu` para sincronizar antes de actualizar y evitar actualizaciones parciales. `install ALL` y los objetos `EBUILD`, `PKG`, `TXZ`, `TBZ`, `SVN`, `CPAN`, `SH`, `LSH` y `COMMAND` no están implementados. Los verbos `fix`, `clean` y `buildtools` pueden tener alcances distintos o faltar según el gestor.

La interfaz sigue el modelo de acciones y objetos explicado en Rosetta, Babel y Packlets. Los packlets actuales son scripts Bash para APT, DNF, Zypper, Pacman, TGZ y las operaciones SummaPack.

Alcance: generar un paquete RPM o TGZ no garantiza que los binarios funcionen en cualquier distribución. El manual señala limitaciones actuales en los nombres de dependencias y las rutas de bibliotecas. Las propuestas de los documentos históricos no deben confundirse con funciones ya implementadas.

Publicar los documentos con subir-doc Para quienes mantienen el proyecto: desde la raíz de las fuentes, comprobar y publicar los PDF con:

```
./subir-doc --prueba
./subir-doc
```

El comando instalado también acepta `subir-doc --doc /ruta/summapack/doc`. Deben existir los archivos locales `summapack.pdf` y `738.pdf`, y el directorio remoto `downloads/summapack/`. Solo se crea su subdirectorio `docs/`. La conexión se configura en `~/.config/simusol/subir` con `SUBIR_SERVIDOR`, `SUBIR_USUARIO` y `SUBIR_DIR`; requiere acceso SSH autorizado. `--prueba` no conecta ni escribe, por lo que no comprueba la existencia del directorio remoto. Los enlaces a los PDF se agregan por separado en TinyMCE.

Herramientas para páginas web La entrada única de SummaPack es `sincronizar-pagina SISTEMA [--prueba]`. Solo se indica el sistema: destinos, rutas, páginas y bases están en la tabla central `share/summapack/data/paginas-web.json`, sustituible por `~/.config/simusol/paginas-web.json`.

Programa	Función	Estado e implementación
`sincronizar-pagina SISTEMA`	Consultar la web, comparar MD/web y coordinar subidas y bajadas de las páginas configuradas.	Implementado: lanzador Bash, configuración JSON, motor Python 3 y transporte SSH. Opción `--prueba` sin escritura.
`subir-pagina`	Publicación directa y opciones avanzadas.	Se conserva por compatibilidad; no hace falta usarlo en el trabajo habitual.
Bajada de páginas	Página CMSimple → HTML local y MD.	Operación interna del comando único; no requiere un programa independiente.
`pandoc`	MD → HTML y HTML descargado → MD.	Dependencia externa escrita en Haskell, invocada como ejecutable; no es código propio de SummaPack.

La web se consulta antes de regenerar el HTML. Solo cambió MD: generar y subir; solo cambió web: descargar e incorporar al MD; cambiaron ambos: detenerse sin sobrescribir. El HTML local no es una tercera fuente editable. La guía de configuración, respaldo y límites está en `doc/SINCRONIZAR-WEB.md` de las fuentes.

Orden de sincronización

```
sincronizar-pagina slex --prueba
sincronizar-pagina slex
```

La tabla inicial configura Slex y la portada de SummaPack para importar desde la web al iniciar; Tabular2 exige revisar la versión inicial. Calcula y Puyuspa están identificados como pendientes de definir fuente y destino. Las subpáginas no listadas y los generadores antiguos no se migran automáticamente.

1. Consultar la web antes de regenerar HTML. Si no se puede leer, detenerse sin sobrescribir archivos locales.
2. Comparar MD y web contra sus respectivas versiones en la última base común, no por fechas.
3. Si cambió solo MD, generar y subir; si cambió solo web, descargar e incorporar al MD; si cambiaron ambos o falta base, detenerse para revisión explícita.
4. Preparar las conversiones y respaldar lo que se reemplazará.

5. Aplicar la actualización. Antes de publicar, verificar bajo bloqueo que la web no haya cambiado desde la consulta; antes de importar, comprobar que el MD no haya cambiado.
6. Actualizar la base local solo después de completar la operación. Ante fallos parciales, revisar las versiones y respaldos antes de reintentar.

La base guarda un par correspondiente de versiones MD y HTML web; no compara directamente ambos formatos. Sin cambios en las fuentes no publica ni modifica el MD, aunque puede recuperar el HTML local si falta. `--prueba` no reemplaza archivos ni crea una base nueva. No hay sincronización periódica automática: se ejecuta por página al invocar el comando. Los generadores antiguos todavía deben conectarse o migrarse a este flujo; no deben ejecutarse antes de consultar la web.

Documentación El manual completo incluye la referencia de todos los componentes: sintaxis, entradas y salidas, requisitos, configuración, efectos sobre archivos o sistemas, ejemplos y límites. Distingue herramientas actuales de las históricas.

Operaciones habituales

Necesidad	Orden o referencia
Consultar versión y sistema	`summapack --version`; `distro +`
Consultar metadatos	`python3 /usr/share/summapack/data/summa.py campo . VERSION`; solo sobre un `.summa` de confianza.
Preparar dependencias	`./preparar --prueba` desde las fuentes; `make dependencias` desde fuentes y `make dependencias-distro` mediante paquetes del sistema.
Simular instalación de dependencias del sistema	`packinstall --prueba UBUNTU libreadline-dev`
Consultar paquetes TGZ instalados	`instalar-tgz --lista`; `instalar-tgz --archivos PAQUETE`
Construir y publicar	`./hacer-pkg` construye; `subir-paquetes` publica; `subir-doc` publica únicamente los PDF.
Integrar lanzadores	`icon_install --prueba /ruta/lanzadores`
Buscar instalaciones que ocultan otras	`findbinlib "interpola" "" "" "" tabular`; con el tercer argumento `BORRA` mueve archivos, no solo informa.

Precauciones: `--prueba` no es una opción universal ni un sandbox. Algunas herramientas pueden consultar la red o crear temporales; `subir-doc --prueba` no conecta. Los `.summa`, las recetas externas y las plantillas pueden ejecutar órdenes. Usar fuentes confiables; las sumas MD5/SHA256 no sustituyen una firma del autor. `templatesumma` se conserva para compatibilidad y puede sobrescribir metadatos; para proyectos nuevos se usa `summa.py`.

Documento	Qué contiene
Manual 2026	Uso actual del archivo `.summa`, construcción, publicación, instalación y hoja de ruta. También disponible en PDF.
Presentacion historica	La presentación de SummaPack conservada en UTUTO, fechada en enero de 2010: objetivos y diseño del proyecto.
Manifiesto Anti-distros	Versión 0.60, de agosto de 2005. Fundamentos de la propuesta de empaquetado desde los proyectos originales.

Documento	Qué contiene
Rosetta Babel y Packlets	Conceptos, objetos, acciones y traducción entre los sistemas de administración de paquetes.
Articulo JSL 2011	Trabajo de Diego Saravia publicado en las 40 JAIIO, Jornadas de Software Libre, páginas 41–59. PDF completo de 19 páginas.

Origen y conservación de los textos Esta sección reúne el manual actual y los documentos históricos conservados con las fuentes del proyecto. Se mantienen sus autores, referencias y avisos de licencia. Los artículos antiguos se identifican como tales; sus enlaces externos y sus instrucciones pueden haber quedado desactualizados.

La instalación, la elección de formatos y las opciones de revisión se mantienen en la guía pública única (`https://www.simusol.org/downloads/0-README-como-instalar`).

M. SummaPack: páginas para CMSimple / TinyMCE

La sección es común a calcula.org, simusol.org y psicro.org. Las fuentes del proyecto permanecen en summapack; no se duplica el software en el repositorio de Calcula. Estos archivos preparan contenido, no cambian el servidor.

M.1. Generar y revisar

Requiere Python 3, Pandoc y los dos PDF de doc/. Si cambió el manual, regenerar primero su PDF con `latexmk -pdf summapack.tex` desde doc/.

```
cd ~/svn/siru/summapack/doc/web
python3 generar.py
python3 verificar.py
```

Abrir `generado/vista/SummaPack.html` en el navegador para recorrer las seis páginas y abrir los PDF locales. Esa vista previa no es el archivo que se pega en TinyMCE.

Página	Nivel CMSimple	Archivo en generado/cmsimple/
SummaPack	Principal	SummaPack.html
Manual 2026	Hija de SummaPack	Manual-2026.html
Presentacion historica	Hija de SummaPack	Presentacion-historica.html
Manifiesto Anti-distros	Hija de SummaPack	Manifiesto-Anti-distros.html
Rosetta Babel y Packlets	Hija de SummaPack	Rosetta-Babel-y-Packlets.html
Articulo JSL 2011	Hija de SummaPack	Articulo-JSL-2011.html

Los nombres de página se mantienen sin tildes ni comas para que sus direcciones coincidan con los enlaces preparados. Los documentos conservan su ortografía. La portada usa H1; cada subpágina usa H2; los títulos internos usan H4–H6, para que CMSimple no los convierta en páginas adicionales.

M.2. Publicar con TinyMCE

1. Guardar una copia del contenido actual del CMS antes de editar.
2. Crear una página principal llamada `SummaPack`.

3. Abrir `generado/cmsimple/SummaPack.html` en un editor de texto y copiar todo el código.
4. En TinyMCE abrir «Código fuente» / «HTML» (`<>`), reemplazar el contenido de esa ventana, aceptar y guardar. No pegar HTML en el editor visual, ni Markdown.
5. Crear las cinco páginas hijas con los nombres de la tabla. En cada una repetir el pegado en «Código fuente» con su archivo correspondiente.
6. Desde la raíz de summapack, ejecutar `./subir-doc --prueba` para revisar y `./subir-doc` para subir los dos PDF a `downloads/summapack/docs/`: `summapack-2026.pdf` y `sumapack-jsl-2011.pdf`. Toma los originales de doc/, no las copias de la vista previa; respalda los documentos que actualiza y no borra otros archivos.
7. Comprobar el menú, los enlaces entre páginas, las tablas, los bloques de código y la apertura de ambos PDF. Si el CMS asigna otra URL a una página, ajustar sus enlaces antes de darla por publicada.

No pegar los seis fragmentos juntos en una página ni reemplazar el archivo global content.htm del sitio.

M.3. Fuentes y conservación

Contenido	Fuente
Portada actual	pagina.md, sincronizada con la página publicada; generar.py la convierte a HTML. inicio.html queda como antecedente y ya no se usa al generar.
Manual web	../summapack.tex; PDF de ../summapack.pdf
Presentación histórica	../UTUTO-SummaPack.html
Manifiesto Anti-distros	../UTUTO-ANTIDISTROS.html
Rosetta, Babel y Packlets	../UTUTO-Rosetta-Babel-y-Packlets.html
Ficha del artículo de JSL	articulo.html; PDF íntegro de ../738.pdf

Se conservan sin modificar los documentos fuente. Del HTML histórico se retiran la plantilla, los menús del sitio y el submenú generado de UTUTO, no los artículos. Se mantienen autores, bibliografía, avisos de licencia y anclas; las imágenes no incluidas en doc/ quedan enlazadas al original, con una advertencia visible. Los enlaces externos se conservan como referencias históricas y no se promete que sigan activos. El PDF de JSL se reproduce completo, no se sustituye por un resumen ni por una extracción de texto que pierda las figuras.

La conversión del manual transforma tabularx en tablas HTML y los títulos en H4–H6; conserva los dobles guiones de las opciones de órdenes y convierte sus citas en enlaces. No ejecuta los comandos del manual. Los archivos de generado/ son resultados reproducibles: editar pagina.md para la portada, articulo.html para su ficha o los documentos originales, no esos resultados. Antes de regenerar la portada, consultar si cambió la web con `sincronizar-pagina summapack --prueba`; si cambió solo la web, importarla primero.

N. Esquema: de nuestras fuentes al usuario, todo con `sp`

Decisiones del autor del 16/11/2026. Todo se hace con `sp`. Lo que `sp` llama por debajo (hacer-pkg, bajar-paquetes, armar-paquetes, subir-paquetes, apt, dnf, podman…) es interno: no se documenta para el usuario ni se le pide que lo llame.

N.1. El recorrido

```
NUESTRAS FUENTES  (svn: el .summa, el MANIFIESTO, los archivos del proyecto)
      |
      |  sp manifiesto    el MANIFIESTO al dia con el arbol
      |  sp paquete       arma las fuentes publicables y el .summa con version
      v
FUENTES PUBLICABLES  NOMBRE_VERSION.tar.gz  (capa automake adentro, configure
      |                                      hecho, .summa y MANIFIESTO adentro)
      |
      +--- sp fuentes    fuentes por FORMATO, para empaquetadores de distros:
      |                  .orig.tar.gz (comun), .dsc + .debian.tar.xz (deb),
      |                  .src.rpm (rpm)
      |
      +--- sp binarios   binarios por DISTRIBUCION, cada uno en su Podman:
      |                  ubuntu, fedora, suse, ututo2027
      v
SERVIDOR DE DESCARGAS
      |  sp subir        sube, como inestable
      |  sp estable      lo subido pasa a estable
      v
EL USUARIO
         sp install ...  o  sp --para fuentes ...
```

N.2. Las tres capas

Capa	Qué es	¿La editamos?	¿Se versiona?	¿Viaja?
summapack	el `.summa`, el `MANIFIESTO` y los archivos del proyecto	sí: es lo único que se edita	sí	sí, adentro del tar.gz
automake	`instalar.am`, `dist.am`, `configure`, `Makefile.in`...	nunca: la genera `sp`	no	sí
deb / rpm	`debian/control`, `rules`...; el `.spec`	nunca: la genera `sp`	no, salvo `debian/changelog`	sí

El detalle y las reglas están en el README, sección "Las tres capas".

N.2.1. La documentación se instala hecha

Decisión del autor del 16/11/2026. Lo que el `.summa` manda a `usr/share/doc/simusol/...` —lo que lista el menú Documentos— se convierte al construir, y se instala sólo lo generado:

Fuente	Se instala	Con
cada `.md`	su `.html`, con los enlaces a otros `.md` pasados a `.html`	`pandoc`; sin él, el texto dentro de un HTML, con un aviso

Fuente	Se instala	Con
cada `.tex` con `\documentclass`	su `.pdf`	el `Makefile` de su carpeta si el `MANIFIESTO` lo trae (figuras que se generan antes, `latexmk`); si no, `pdflatex` dos veces
los `.tex` sin `\documentclass`	nada: son partes del principal y van dentro del PDF	

Lo hace `summa-automake.sh`, en el `instalar.am`: ningún proyecto tiene que escribirlo. Es la capa automake: el `.md` y el `.tex` siguen en las fuentes, lo generado no se versiona y va dentro del paquete. Por eso cada proyecto con documentación pide `pandoc`, y texlive si tiene LaTeX, en sus `BUILDDEPENDS`.

N.3. Qué hay en el servidor

Qué	Para quién	Cuántos	Lo arma
el `sp` autónomo, con summapack adentro, y la guía `0-README-como-instalar`	todos: es la puerta de entrada	uno	`sp subir --bootstrap summapack`
`PROYECTO.summa`	`sp`, para saber la versión estable y las dependencias	uno por proyecto	`sp paquete`
`PROYECTO/NOMBRE_VERSION.tar.gz`	Instalación típico: lo compila `sp` en su máquina	uno por proyecto, sirve para todas las distribuciones	`sp paquete`
binarios por distribución	quien prefiera no compilar	uno por proyecto y por distribución	`sp binarios`
fuentes por formato (`.dsc`, `.src.rpm`), en `PROYECTO/fuentes-distro/VERSION/`	un empaquetador de una distro	uno por proyecto y por formato	`sp fuentes`; las sube `sp subir --tipo distro`
la ISO de Ututo 2027	quien instale la distribución entera	una	`sp iso ututo2027`

Cada archivo va con su `.md5`.

No hace falta repositorio apt ni dnf: `sp` resuelve las dependencias. Las de la distribución las instala con el gestor de la máquina, traduciendo los nombres; las nuestras las baja y las construye o instala igual que el proyecto pedido.

Compilar no es una carga para el usuario: quien usa sceptre ya compila Fortran en cada simulación, así que las herramientas tienen que estar igual.

N.4. Las fuentes van por formato; los binarios, por distribución

	Separación	Ejemplos
Fuentes	por **formato**: una del programa y una por formato de empaquetado	`simusol_V.orig.tar.gz`, `simusol_V-N.dsc`, `simusol-V-N.src.rpm`
Binarios	por **distribución**: cada una compila contra sus propias bibliotecas, y la marca va en el nombre	`simusol_V_all.deb` (Ubuntu, Debian), `simusol_V+ututo2027_all.deb`, `simusol-V.fedora.noarch.rpm`, `simusol-V.suse.noarch.rpm`, `simusol_V+arch_all.tgz`

Un mismo `.src.rpm` sirve para Fedora y para openSUSE: las dependencias que se llaman distinto quedan escritas como alternativas, por ejemplo `(gcc-gfortran or gcc-fortran)`.

N.5. Lo que hacemos nosotros

Con todo comiteado, parado en el árbol o dando `LUGAR/NOMBRE`.

N.5.1. 1. Mantener las fuentes

```
sp manifiesto LUGAR/NOMBRE     # genera el MANIFIESTO, o sincroniza svn con él
```

La regla vigente para archivos nuevos y exclusiones se describe en la sección 5.

N.5.2. 2. Armar

```
sp paquete LUGAR/NOMBRE                    # NOMBRE_VERSION.tar.gz y NOMBRE_VERSION.
    summa
sp fuentes LUGAR/NOMBRE [SALIDA]           # .orig.tar.gz, .dsc, .debian.tar.xz, .src
    .rpm
sp binarios LUGAR/NOMBRE [DISTRO...]       # ubuntu, fedora, suse, ututo2027; sin
    ninguna, las cuatro
sp iso ututo2027                           # la ISO de Ututo 2027, desde nuestras
    fuentes
```

`sp iso ututo2027` arma la ISO como la instalaría el usuario: publica nuestras fuentes (las de `sp paquete`) con la forma del servidor, arma la raíz de Trisquel y, adentro, instala cada proyecto con `sp --para fuentes-paquete install` (make install en un directorio, un `.deb` con eso, y el paquete instalado). Las fuentes quedan en la ISO, en `/usr/src/ututo2027`. Con `--preparar`, `--reanudar`, `--solo-iso` o `--disco-vm` hace sólo esa etapa.

`sp binarios` usa las fuentes que dejó `sp paquete`, así que va después. Cada distribución compila en su propio Podman, con la cadena completa de nuestras dependencias. Los paquetes que se arman en la máquina propia no se suben nunca: un `.deb` de Ubuntu 24.04 no anda en 22.04.

N.5.3. 3. Publicar

Comitear no autoriza a subir: cada subida la decide el autor.

```
sp subir --prueba LUGAR/NOMBRE             # dice qué subiría, sin subir
sp subir LUGAR/NOMBRE                      # sube fuentes, .summa y binarios, como
    inestable
sp subir --tipo fuentes LUGAR/NOMBRE       # sólo las fuentes y el .summa
sp estable LUGAR/NOMBRE                    # lo inestable pasa a estable
```

```
sp subir --bootstrap LUGAR/summapack        # la raíz: el sp autónomo, la guía y los
    catálogos
sp subir --tipo distro LUGAR/NOMBRE         # las fuentes por formato, a PROYECTO/
    fuentes-distro/VERSION/
```

Las fuentes por formato van aparte y tal cual, sin pasar por inestable ni estable: el `.dsc` nombra al `.orig.tar.gz` y al `.debian.tar.xz` por su nombre exacto, así que no se los puede renombrar, y cada versión queda en su carpeta. Si no están armadas, `sp subir --tipo distro` corre antes `sp fuentes`.

Si una distribución ya tenía binarios de un proyecto en el servidor y faltan los de la versión nueva, `sp subir` los arma antes de subir.

N.5.4. Orden de un lanzamiento completo

1. `sp paquete` de cada proyecto, empezando por summapack y siguiendo por sus dependencias: unidades, exprtk, slex, tabular2, simplot, siru, lanzador, sceptresimusol, simusol, dia-simusol.
2. `sp binarios` de cada uno.
3. `sp subir --prueba` de cada uno; revisar.
4. `sp subir` de cada uno.
5. Probar la instalación en una máquina limpia (un Podman nuevo): `sp --para fuentes install simusol`.
6. `sp estable` de cada uno.
7. Si cambió summapack, `sp subir --bootstrap summapack`.

N.6. Lo que hace el usuario

N.6.1. Sin nada instalado

```
curl -fsSLO https://www.simusol.org/downloads/sp
sh sp --para fuentes-paquete simusol
```

El `sp` autónomo instala summapack y después el proyecto con todas sus dependencias.

N.6.2. Con sp instalado

Para	Orden
instalar directamente desde las fuentes (make install en `/usr/local`, con lista para desinstalar)	`sp --para fuentes install simusol`
lo mismo, en el home y sin root	`sp --para fuentes --prefijo ~/.local install simusol`
forma preferida cuando está disponible: compilar acá e instalar como paquete de la distribución	`sp --para fuentes-paquete install simusol`
instalar el binario de esta distribución	`sp install simusol`
un binario de una distribución en particular	`sp --para ututo2027 install simusol` (o `deb`, `fedora`, `suse`, `arch`)
ver qué haría, sin hacerlo	`sp --ensayo ...`
desinstalar	`sp remove simusol`
lo instalado desde fuentes	`sp list-source`
buscar, informar	`sp search TEXTO`, `sp info PAQUETE`

Para	Orden
actualizar todo	`sp install WORLD`

En Ututo 2027, `sp` es el gestor de paquetes de la distribución: ver `siru/ututo/GESTOR-DE-PAQUETES.md`.

N.6.3. Desde el árbol de svn, sin internet

```
svn/siru/summapack/sp/sp                          # instala o actualiza summapack del
    árbol
sp --arbol --para fuentes install simusol         # un proyecto desde el árbol, no
   del servidor
```

N.7. Lo que hace un empaquetador de otra distro

No necesita summapack. Toma las fuentes por formato y sigue con las herramientas de su distribución:

```
dpkg-source -x simusol_V-N.dsc && cd simusol-V && dpkg-buildpackage -b    # deb
rpmbuild --rebuild simusol-V-N.src.rpm                                    # rpm
tar xzf simusol_V.orig.tar.gz && cd simusol-V && ./configure && make      # cualquiera
```

N.8. Lo que todavía falta

- **Armar los paquetes nuevos.** Las fuentes armadas que hay en el árbol son de r4443 a r5350, anteriores a todo lo de noviembre.
- **Probar `sp binarios`**, que todavía no armó ningún binario, en particular los de ututo2027.
- **La ISO de Ututo 2027** no tiene orden de `sp` para subirla.
- **Los binarios no salen de las fuentes por formato**, sino de las fuentes publicables: cada Podman debería construir desde el `.dsc` o el `.src.rpm`.
- **lanzador** no arma mientras no se declaren los archivos nuevos de `corpus/originales/`.
- Los pendientes de cada capa están en el README, "LO QUE TODAVIA NO CUMPLE".

Ñ. Paquetes: del .summa al servidor de descargas

Pedido del autor, 07/10/2026: que cada proyecto salga como tar.gz, tgz, deb y rpm, con sus md5, que haya como subirlos y marcar la ultima version, y que summapack sea la base comun de todos. "El summa es la fuente: si alguien cambia el summa cambia todo."

Ñ.1. Guia rapida

Guía pública única: `https://www.simusol.org/downloads/0-README-como-instalar`. El nombre público actual es `sp`; reemplaza a `installsumma.sh`. Las menciones de este último en pruebas y relatos fechados se conservan como historia, no como instrucciones vigentes. `bajar-paquetes` es un motor interno.

El nombre vigente es `hacer-pkg`, con la parte común `hacer-pkg.inc`: construye DEB, RPM, TGZ y fuentes `.tar.gz`. Los nombres anteriores `hacer-deb` y `hacer-deb.inc` quedan como enlaces de compatibilidad, no como copias independientes. Los scripts de los proyectos activos de Siru y Calcula y el instalador de Siru usan el nombre nuevo; no se cambian las versiones históricas ni los paquetes ya publicados. Para una copia aislada de Calcula se necesita una versión de SummaPack que incluya `hacer-pkg.inc`, o indicar con `SUMMAPACK_DATA` la copia actual de las fuentes.

Ñ.1.1. Publicar una version nueva

1. Cambiar el codigo y, si cambia algo del paquete (dependencias, opciones), el `.summa` del proyecto.

2. Subir la version en el .summa (VERSION="6.0-39") y anotar el cambio en `debian/changelog` del proyecto (`dch -v 6.0-39`).

3. Armar: `PROYECTO/hacer-pkg` (por ejemplo `~/svn/siru/tabular2/hacer-pkg`). Deja tar.gz, deb, tgz, rpm, el .summa con version y los .md5, y los anota en `subir.lista` de la raiz del repositorio.

4. Subir lo armado (queda como inestable, en `PROYECTO/otros/`):

   ```
   subir-paquetes $(cat ~/svn/siru/subir.lista)
   subir-paquetes $(cat ~/svn/calcula/subir.lista)
   ```

   Lo que ya esta en el servidor con el mismo md5 no se vuelve a subir.

5. Probarlo bajando la inestable: `bajar-paquetes --inestable PROYECTO`.

6. Cuando esta bien, marcarlo estable: `subir-paquetes --estable PROYECTO`. Los paquetes pasan a `PROYECTO/`, los estables anteriores vuelven a `PROYECTO/otros/` y el UNSTABLE desaparece (`--inestable` baja entonces la estable).

`--prueba` existe en los publicadores y el descargador, no en `hacer-pkg`, `dch` ni todas las órdenes del flujo. Su alcance varía: `subir-doc --prueba` no conecta; `subir-paquetes` puede consultar por SSH, y el arranque/flujo desde fuentes puede descargar metadatos y crear temporales. No es un sandbox.

Ñ.1.2. Instalar en otra maquina

Con SummaPack instalado:

```
sp install calcula
```

Desde una máquina sin SummaPack:

```
curl -fsSLO https://www.simusol.org/downloads/sp
sh sp --para deb calcula
```

Una sola orden instala SummaPack, el proyecto y sus dependencias. Para fuentes e instalación directa en `/usr/local`, elegir `--para fuentes`; para construir paquetes nativos, `--para fuentes-paquete`. Son alternativas. Las instrucciones completas y `--prueba` se mantienen en la guía canónica (`https://www.simusol.org/downloads/0-README-como-instalar`).

La publicación del arranque se realiza con `sp subir --bootstrap summapack`; publicar es una operación de mantenimiento distinta de instalar.

Ñ.1.3. Preparar el acceso al servidor (hecho el 07/10/2026)

En simusol.org (es la misma maquina que ututo.org, cada sitio con su propio downloads/), www-data no puede entrar por ssh: intentar subir con el hace que fail2ban bloquee la IP (se desbloquea en el servidor con `fail2ban-client set sshd unbanip IP`). Se publica con un usuario propio, del grupo www-data, que solo puede escribir en downloads/. En el servidor, como root:

```
adduser --disabled-password --gecos '' publicador
usermod -aG www-data publicador
chgrp -R www-data /var/www/simusol.org/public_html/downloads
chmod -R g+w /var/www/simusol.org/public_html/downloads
chmod g+s /var/www/simusol.org/public_html/downloads
mkdir -p /home/publicador/.ssh
cp /root/.ssh/authorized_keys /home/publicador/.ssh/
chown -R publicador:publicador /home/publicador/.ssh
chmod 700 /home/publicador/.ssh; chmod 600 /home/publicador/.ssh/authorized_keys
```

Y en la maquina desde donde se sube, `~/.config/simusol/subir`:

```
SUBIR_SERVIDOR=simusol.org
SUBIR_USUARIO=publicador
SUBIR_DIR=/var/www/simusol.org/public_html/downloads
```

Lo leen subir-paquetes, bajar-paquetes y `sp subir --bootstrap summapack`; lo que venga en el entorno manda (para probar contra un directorio local: `SUBIR_SERVIDOR=local SUBIR_DIR=/tmp/x`).

El 07/10/2026 se paso todo lo que habia en downloads/ a downloads/viejos/ (`subir-paquetes --reorganizar`); lo nuevo va por proyecto.

Ñ.2. Otras distribuciones: el problema

Los paquetes se compilan en Ubuntu 24.04. Un binario compilado ahi pide las bibliotecas de ese sistema con su version, y no siempre anda en otro:

- **libc**: lo compilado pide libc6 >= 2.38. Debian 12 trae 2.36: no instala.
- **Versiones de bibliotecas**: calcula usa gsl 2.7 (libgsl.so.27); Fedora 44 y openSUSE Tumbleweed traen gsl 2.8 (libgsl.so.28): no corre aunque se instale. Lo mismo puede pasar con suitesparse (KLU, Mongoose).
- **Nombres de paquetes**: libc6 en Debian es glibc en Fedora y SUSE; libreadline8t64 es readline, libreadline8... Se resuelve con el diccionariodepaquetes (los .rpm piden glibc y, para las bibliotecas, la biblioteca misma: libreadline.so.8()(64bit)).
- **Lugar de las bibliotecas**: van en /usr/lib/x86_64-linux-gnu, que es de Debian; Arch no la mira. instalar-tgz la agrega a /etc/ld.so.conf.d.

Probado el 07/10/2026 en contenedores, con `sh installsumma.sh tabular2`:

sistema	resultado
Ubuntu 24.04	instala y anda (con gnuplot)
Debian 13	instala y anda
Debian 12	no instala: libc6 2.36 < 2.38, y no tiene libreadline8t64
Fedora 44, openSUSE Tumbleweed	con los nombres viejos (libc6) no instalaba; corregido con el diccionario
Arch	instalaba sin dependencias y sin encontrar las bibliotecas; corregido

La solucion: compilar en cada distribucion. En `siru/summapack/distros` estan los entornos (contenedores con podman) para compilar los paquetes en la version actual de openSUSE, Fedora y Arch (`summapack/distros/compilar fedora`), cada vez que aparece uno de estos problemas. Salen paquetes propios de cada una, con su marca

(`tabular-2.0-15.fedora.x86_64.rpm`, `.suse`, `+arch`; hasta el 08/10/2026 eran `.fc44` y `.tw`), que conviven en el servidor con los de Ubuntu; bajar-paquetes baja los que se le piden con `--para` (o se eligen del menu). Ver siru/summapack/distros/README.md. Pedido del autor, 07/10/2026.

Pero no es estable (por eso, desde el 08/10/2026, un solo juego compilado en Ubuntu 22.04: ver "Un solo juego de paquetes"). Cada paquete sirve para la version de la distribucion en que se compilo. Fedora saca una version nueva cada seis meses, y openSUSE Tumbleweed y Arch cambian todo el tiempo: con las bibliotecas nuevas un paquete viejo puede dejar de andar, y hay que recompilarlo y volver a subirlo. Lo mismo vale para los .deb de Ubuntu (en Debian 12 ya no andan). Por eso la unica forma estable de hacer paquetes es con las fuentes (`--para fuentes`, ver "Desde las fuentes"): se compilan en la maquina, con lo que esa maquina tiene. Los paquetes binarios son una comodidad para las distribuciones de hoy.

Con los paquetes compilados alli, `sh installsumma.sh tabular2` instala y anda (con gnuplot) en Fedora 44, openSUSE Tumbleweed, Arch y Ubuntu 24.04, cada uno con los suyos.

Ñ.3. De donde viene

La idea esta en `doc/`:

- `doc/738.pdf`: "Sumapack: herramientas para la compilacion, empaquetado e instalacion de software libre en diferentes arquitecturas, sistemas operativos y distribuciones GNU/-Linux", D. Saravia, 40 JAIIO - JSL 2011.
- `doc/UTUTO-ANTIDISTROS.html`: el Manifiesto Anti-distros (2005).

Lo que aca se hace es lo que proponen: el empaquetado vive en el proyecto original y no en cada distribucion ("make deb, make rpm, tgz"); los archivos de cada sistema de paquetes (control, .spec, .lsm, .ebuild) "se generan desde un archivo comun", que es el `.summa`; y un deb y un rpm difieren solo en el formato: el .rpm y el .tgz se arman desde el mismo arbol que el .deb.

Lo que el articulo describe y todavia no esta en esta forma nueva, como hoja de ruta:

del articulo	hoy
Rosetta (rosetta.dat): el nombre de cada paquete en cada distribucion	`diccionariodepaquetes`; lo usa el .rpm para traducir dependencias
.lsm y .ebuild desde el archivo comun	solo debian/ y el .spec del rpm
`lacks` y `make prepare`: instalar lo que falta para compilar	hecho: `make dependencias` y el aviso de hacer-pkg
Uinstall: instalar un .tgz llevando registro, para desinstalarlo	hecho: `instalar-tgz`
Bootstrap: un script que instala summapack y sus dependencias	hecho: `sp`, que incorpora el motor interno de descarga
dependencias por lo que se provee y requiere (bibliotecas), no por nombres	`${shlibs:Depends}` en el .deb; el .rpm tiene AutoReqProv apagado

Ñ.4. Summapack, la base comun

Un solo proyecto (este directorio), un solo `summapack.summa`, y tres paquetes:

paquete	que trae	arquitectura
`summapack`	makefile.lib, diccionariodepaquetes, los scripts de bin/	all

paquete	que trae	arquitectura
`libmem-util0`	mem_util, la biblioteca C de base (memoria, archivos, rutas, fechas)	any
`libmem-util-dev`	sus encabezados, el .a, el enlace de desarrollo y el mem_util.pc	any

`summapack` depende de `libmem-util-dev`: alcanza con pedir summapack para tener las dos cosas.

La única fuente vigente de mem_util está en `summapack/mem_util/` dentro del repositorio Siru. Se incorporó con su historia SVN en r2473 y la copia antigua de la raíz se retiró el 09/10/2026, tras comprobar que no tenía código ni documentación exclusivos. Su empaquetado separado se conserva en el historial de SVN. Los paquetes siguen llamándose `libmem-util0` y `libmem-util-dev`; los producidos por SummaPack reemplazan a los antiguos (2017.2.1.R1008-2 > 1.0-2).

El instalador general construye SummaPack antes de los demás proyectos y obtiene de él la biblioteca. `instalar mem_util` se conserva como alias de `instalar summapack`, también en modo `--fuentes`. Los cambios en la biblioteca se hacen únicamente aquí. `Makefile.am` ya no incluye en `EXTRA_DIST` un descriptor independiente inexistente: el descriptor es el `summapack.summa` del directorio padre.

Verificación de la retirada: comparados todos los archivos versionados de ambas copias; el código, las pruebas y la documentación eran idénticos. Lo exclusivo era el empaquetado independiente y su referencia en `EXTRA_DIST`. Una exportación limpia de `summapack/mem_util` pasó `make check` y `make distcheck`, incluyendo instalación y desinstalación temporales. El instalador general se comprobó con las recetas de fuentes simuladas: selección por `summapack`, alias `mem_util` y construcción de la base antes del resto. No se modificaron las bibliotecas instaladas del sistema.

Ñ.5. El .summa

Cada proyecto tiene en su directorio un `NOMBRE.summa` con todo lo necesario para armar sus paquetes: la version completa (VERSION="2.0-15"), resumen y descripcion, dependencias, lo que hace falta para compilar, las opciones de configure y de dh, el resto de debian/rules, los paquetes que salen y los demas archivos de debian/ (copyright, *.install,* .docs...). Es un archivo de shell; sus campos se describen en esta seccion.

Un .summa puede generar varios paquetes. `PACKAGES` los lista; el primero es el principal y usa SUMMARY, DESCRIPTION y DEPENDS. Cada uno de los otros tiene los suyos con el nombre del paquete adelante (`libmem_util_dev_SUMMARY`, con `_` en lugar de `-`, `.` y `+`), y todos `NOMBRE_ARCH` y `NOMBRE_EXTRA` (otros campos de debian/control: Section, Multi-Arch, Suggests...). Que archivo va a cada paquete lo dice su `ARCHIVO_NOMBRE_install`. Si una ruta existe tambien en el arbol fuente (como `etc` en summapack), ponerla como `debian/tmp/etc`: dh_install busca primero en el arbol fuente.

En `debian/` queda solo `changelog`: es la historia, con lo que cambio en cada version. Su ultima entrada tiene que ser la VERSION del .summa (si no, hacer-pkg lo dice: `dch -v VERSION`); BASEVERSION, la parte de arriba (2.0), queda para las herramientas viejas. Para sacar una version nueva: cambiar VERSION en el .summa y agregar la entrada en el changelog. Todo lo demas de debian/ lo genera `summa.sh debian` en la copia limpia que arma hacer-pkg; no se edita ni se versiona.

```
bash summa.sh crear DIR        un .summa nuevo desde un debian/ completo
bash summa.sh actualizar DIR   pasa un .summa viejo de makefile.lib al formato nuevo
```

```
bash summa.sh debian DIR       genera DIR/debian/ (para mirarlo, o para
                           correr dpkg-buildpackage a mano)
bash summa.sh campo DIR CAMPO  un valor
```

Para cambiar algo de un paquete (una dependencia, una opcion de configure): editar el .summa, agregar una entrada en debian/changelog (`dch -i`) y correr hacer-pkg. Un .summa nuevo hay que agregarlo a svn (`svn add`): hacer-pkg arma desde `svn export`, y lo que no esta agregado no entra.

Ñ.6. hacer-pkg

`PROYECTO/hacer-pkg` arma desde una copia limpia y deja junto al proyecto (en la raiz del repositorio, para calcula y puyuspa):

archivo	que es
`NOMBRE_VERSION.tar.gz`	las fuentes, reproducible (mismo contenido, mismos bytes); trae `configure` ya hecho
`PAQUETE_VERSION_ARQ.deb`	el paquete Debian/Ubuntu, uno por paquete del .summa
`PAQUETE_VERSION_ARQ.tgz`	lo mismo que el .deb, para desempaquetar en / en cualquier distribucion; lleva ademas `var/lib/summapack/tgz/PAQUETE.control` (nombre, version, dependencias)
`PAQUETE-VERSION.ARQ.rpm`	el mismo arbol como rpm (necesita `sudo apt install rpm`; las dependencias se traducen con diccionariodepaquetes)
`NOMBRE_VERSION.summa`	el .summa con que se armo
`*.md5`	uno por archivo

Todos estan en `svn:ignore`.

Ñ.7. Interfaz de construcción e instalación: objetivo y estado

La interfaz buscada para cada proyecto activo es la siguiente. La tabla es un **contrato por completar**, no una afirmación de que todos los paquetes lo cumplan hoy:

regla	qué hace
`make`	lo que haya que construir; en los que no compilan, nada
`make install`	instala bajo `prefix` (o `PREFIX`), respetando `DESTDIR`
`make uninstall`	revierte una instalación manual hecha con los mismos `prefix`/`PREFIX` y `DESTDIR`; no se usa para quitar paquetes nativos
`make dist`	crea el archivo de fuentes distribuible sin instalar ni modificar el árbol de trabajo
`make deb`, `make rpm`, `make tgz`, `make pkg`	los paquetes, con `hacer-pkg`
`make lista`	lo que instalaría, sin instalar (en los que no usan automake)
`make check`	las pruebas (lo que corre el paquete en su `dh_auto_test`)

Auditoría del 12/10/2026: Calcula, Unidades, Slex y Tabular2 tienen los objetivos de Autotools `install`, `uninstall` y `dist`. Simplot, Siru, ExprTk y Puyuspa tienen los tres objetivos manuales. Dibuja incorporó `dist` en r3083. SummaPack incorporó `uninstall` con manifiesto de archivos y huellas en r3082; su `dist` heredado usa ahora `crear-fuentes`, sin modificar el árbol. El ciclo `install`/`uninstall` de SummaPack se probó con `DESTDIR` en una exportación limpia y dentro de Debian 13, Ubuntu 24.04, Fedora 44 y openSUSE Tumbleweed. `make dist` de Dibuja produjo el tarball en una exportación limpia. Esto verifica esos objetivos, **no** el contrato completo en cada proyecto: faltan ciclos equivalentes para los demás y una interfaz uniforme `make deb/rpm/tgz/pkg` (varios siguen usando `./hacer-pkg`).

El recorrido de la lista lo hace `bin/instalar-del-summa`, no cada Makefile. Lee un bloque de archivos del `.summa` —la misma lista con la que se arma `debian/install` para el `.deb`— e instala o desinstala:

```
instalar-del-summa --destino DIR --quitar PREFIJO [--mapear V=N]
                   [--bloque NOMBRE] [--lista] [--desinstalar]
```

opción	para qué
`--destino`	la raíz, con el `DESTDIR` y el prefijo ya adentro
`--quitar`	lo que hay que sacarle al destino que trae el `.summa`, que es el del paquete: `usr/`, `var/www/simusol.org/public_html/`
`--mapear V=N`	cambiar un destino: siru pone sus módulos en `lib/` y no en `share/perl5`, porque `bin/siru` los busca con `use lib "$FindBin::Bin/../lib"`
`--bloque`	cuál lista: `ARCHIVO_install`, `ARCHIVO_siru_manpages`, `_docs`, `_examples`

Los permisos los pone él: 755 lo que está bajo `bin/` o termina en `.pl`, `.sh` o `.py`, y 644 el resto, que es lo que hacen los paquetes en su `override_dh_fixperms`. Así la lista está una sola vez (en el `.summa`) y el recorrido también (acá), y el `.deb` y el `make install` no se pueden desincronizar.

Lo usan los `make install` de puyuspa, siru y exprtk, que lo buscan como `hacer-pkg` busca su `.inc` (`SUMMAPACK_DATA`, la copia de trabajo al lado, lo instalado) y, si no está, lo dicen y no siguen. **Que todos los paquetes dependan de summapack es deliberado** (decisión del autor, 08/10/2026): es lo que el instalador pone primero, y ningún paquete nuestro es independiente de todo.

En los Makefile manuales, `prefix` y `PREFIX` son equivalentes. En Autotools se fija el prefijo al configurar con `./configure --prefix=...`. Para una instalación **manual**, `DESTDIR` prepara una raíz temporal:

```
make install prefix=/opt/simusol DESTDIR=/tmp/prueba
make install PREFIX=/opt/simusol        # lo mismo
```

El arranque público `sh sp --para fuentes PROYECTO` compila e instala mediante `make install` en `/usr/local`, admite `--prefijo` y registra los archivos para `sp remove PROYECTO`. La alternativa `sh sp --para fuentes-paquete PROYECTO` construye paquetes nativos `.deb` o `.rpm` en la máquina destinataria y los instala con su gestor. El formato se elige según la guía canónica; no se deben confundir ambos recorridos. Los proyectos sin Autotools siguen requiriendo recetas de construcción propias cuando se empaquetan:

paquete	por qué es distinto
puyuspa	no compila nada: copia páginas y CGI al directorio que publica el sitio (`sitio=`, y con `prefix` va a `$prefix/share/puyuspa`)
siru	es Perl: ejecutables, `lib/Siru`, manuales, `.md` y ejemplos
exprtk	es una cabecera y su `.pc`

Lo que **no** se unificó, a propósito: la implementación de adentro. Forzar autotools donde no hay nada que compilar no ayuda —puyuspa tenía un `Makefile.am` que nadie usó nunca, justamente por eso— y lo que importa es que la interfaz sea la misma, no el andamio.

Ñ.7.1. `make distcheck` es la prueba de todo esto (08/10/2026)

En los cinco paquetes con autotools —calcula, mem_util, unidades, tabular2 y slex— **`make distcheck` pasa**, y conviene que siga pasando, porque prueba tres cosas de una: que `make install` y `make uninstall` están completos (instala en un prefijo temporal y comprueba que al desinstalar no quede nada), que el tarball lleva todo lo necesario para rehacerse, y que las pruebas no suponen que el srcdir y el directorio de construcción son el mismo.

Las dos cosas que destapó la primera vez que se corrió en todos:

- **El `.summa` tiene que estar en `EXTRA_DIST`.** En cuatro paquetes no estaba, o estaba con un nombre que no existe (`mem_util.summa` por `mem-util.summa`, `tabular.summa` por `tabular2.summa`), así que `make dist` dejaba afuera la única fuente de `debian/`, del `.spec` y de las dependencias: de ese tarball no se podía rehacer el paquete aunque compilara. Arreglado en r2602, r2604 y calcula r243.
- **Las pruebas tienen que buscar sus datos en `$srcdir`** y sus ejecutables en el directorio actual, que fuera del árbol no son el mismo. Cuatro casos arreglados en r2603; el detalle, en `calcula/PENDIENTE-instaladores.md`. El patrón que quedó, para cuando aparezca otro: copiar los datos al build en una regla colgada de `check-am`, sacarlos en `distclean-local` (si no, `distcleancheck` se queja de archivos que sobran) y pasar el srcdir por `AM_TESTS_ENVIRONMENT` cuando lo necesita un script.

Los paquetes con Makefile manual necesitan una prueba equivalente de `install`/`uninstall` con un prefijo temporal y una prueba adicional de `dist`; el instalador automático ya no sustituye esa prueba.

Ñ.8. El servidor de descargas

Organización por proyecto pedida el 07/10/2026, con el arranque público actualizado:

```
downloads/
  viejos/              todo lo que habia antes
  0-README-como-instalar  guía pública única
  sp                   instala SummaPack, proyectos y dependencias
  diccionariodepaquetes  los nombres de cada paquete en cada distribucion
  calcula.summa        el .summa estable de cada proyecto, con la
                       version estable (VERSION="6.0-38")
  tabular2.summa ...
  calcula/             los paquetes estables (.deb .rpm .tgz .tar.gz)
    otros/             todo lo demas: .md5, inestables, versiones
                       anteriores, .summa con version, y mientras haya
                       algo inestable, PAQ_UNSTABLE... (enlace)
  tabular2/            libtabular2-0, libtabular2-dev y tabular: el
```

```
   otros/             directorio es el del proyecto (del .summa)
 comunes.summa        el proyecto "comunes" (08/10/2026): PACKAGES y
                      VERSION de los simusol-*
 comunes/             simusol-readline, -pcre2, -gsl, -openblas y
                      -suitesparse (.deb .rpm .tgz): las
                      bibliotecas de afuera, compiladas en Ubuntu 22.04
   fuentes/           copia de las fuentes de fuentesexternas (gsl,
                      OpenBLAS, cmake, autoconf...), con SHA256SUMS
   otros/
```

Ñ.9. sp subir (18/10/2026)

Pedido del autor: subir un proyecto con una sola orden, que solo funcione desde donde esta su .summa y con todo comiteado:

```
cd ~/svn/calcula/calcula-6.0 && sp subir calcula
sp subir ~/svn/calcula/calcula-6.0/calcula      (desde otro lado)
```

Controla que este NOMBRE.summa en esa carpeta, que sea una copia de trabajo de svn y que no haya cambios sin comitear (`svn status`; lo que svn no conoce no cuenta); si no, no sube nada y dice que falta. Arma con `./hacer-pkg` (desde lo comiteado) el tar.gz y el .summa con version, los sube con subir-paquetes como UNSTABLE. Tres ordenes:

```
sp paquete calcula    arma (hacer-pkg): fuentes, .summa y paquetes; con
                      cambios sin comitear avisa pero arma (para probar)
sp subir calcula      sube lo armado, como UNSTABLE (exige todo
                      comiteado); si no esta armada la version actual,
                      llama antes a sp paquete
sp estable calcula    lo subido pasa a estable (no hace falta estar en
                      la carpeta)
```

`--prueba`: dice que haria.

Que sube `sp subir`: sin `--tipo`, todo lo de esa version, de todos los paquetes del .summa: fuentes (tar.gz), .deb, .rpm y .tgz. Con `--tipo` (fuentes, deb, rpm, tgz; separados por coma: `--tipo rpm`, `--tipo deb,rpm`), solo eso. El .summa va siempre. Los binarios salen solo de lo armado en las bases viejas, en `summapack/distros/salida/viejas/DISTRIBUCION/` (o `$SP_BINARIOS`); lo que hacer-pkg arma en la maquina del autor no se sube (decision del autor, 18/10/2026: un .deb de Ubuntu 24.04 no anda en 22.04 ni en Debian 12). `publicar-cadena` tiene que dejar ahi lo que arma.

Antes de subir, `sp subir` compara los requisitos versionados del `.summa` (`PROYECTOS_REQUERIDOS`, `BUILDDEPENDS` y `DEPENDS`) con los `.summa` estables del servidor. Si una dependencia de Simusol está atrasada, busca su fuente local, comprueba la versión mínima, la sube y la estabiliza primero; después vuelve al proyecto pedido. Si ese proyecto ya ofrecía binarios, arma en los contenedores antiguos los formatos que falten para la nueva versión. `sp estable` vuelve a comprobar las dependencias y rechaza una promoción incompleta. Las dependencias externas siguen a cargo del gestor de paquetes de cada distribución.

Adonde sube: el campo `SUBIR_A` del .summa, la cadena completa `usuario@servidor:/camino` (pedido del autor, 18/10/2026):

```
SUBIR_A="publicador@simusol.org:/var/www/simusol.org/public_html/downloads"
```

En svn va el destino real: el grupo de desarrollo (los que participan del svn) es de confianza; lo que no tiene es la clave ssh del servidor, y sin ella no se sube. En lo publicado (el tar.gz y el NOMBRE_VERSION.summa) hacer-pkg deja la plantilla `SUBIR_A="usuario@servidor:/camino"`; si `sp subir` o `sp estable` la encuentran, no suben nada y piden cambiar SUBIR_A en el .summa. Antes de subir se comprueba que desde esa maquina se entra por ssh a `usuario@servidor`

sin contrasena; si no, dice como habilitarlo (`ssh-copy-id`). Tambien se admite un nombre de `~/.config/sp/destinos` (renglones `NOMBRE usuario@servidor:/camino`) y `SUBIR_A=""` (la configuracion de subir-paquetes, `~/.config/simusol/subir`). `sp estable` tambien necesita el .summa: se usa como `sp subir` (parado en el proyecto, o `LUGAR/calcula`). Los binarios no: salen de las bases viejas. Programa: `bin/sp-subir`.

Cuando `sp subir` tiene que armar (no esta lo de la version actual), arma solo las fuentes: el tar.gz y el .summa con version, con sus .md5 (`HACER_PKG_SOLO_FUENTES=1 ./hacer-pkg`). Los paquetes de esta maquina no se suben y armarlos pedia tener instaladas las dependencias de compilacion (siru no se armaba con un simplot viejo instalado). `sp paquete` arma todo.

Ñ.9.1. El arranque (bootstrap)

Son los archivos que dejan empezar en una maquina que no tiene nada de Simusol, ni siquiera summapack. Van sueltos en la raiz de descargas, fuera de las carpetas de cada paquete:

archivo	que es
`0-README-como-instalar`	la guia: que bajar y que orden correr (el 0 la pone primera en el indice)
`sp`	el sp autonomo: `bin/sp-bootstrap.in` con `bin/bajar-paquetes` adentro (lo arma `distros/armar-sp`)
`diccionariodepaquetes`	el nombre de cada dependencia en cada distribucion
`fuentesexternas`	de donde bajar las fuentes de lo que no es nuestro

Quien empieza baja `sp` y corre, por ejemplo, `sh sp --para fuentes siru`: sp baja el diccionario y las fuentes, pone primero summapack y despues el resto de la cadena.

Se sube con `sp subir --bootstrap summapack` (desde la carpeta de summapack, o `sp subir --bootstrap LUGAR/summapack`; `--prueba` dice que haria), adonde diga el `SUBIR_A` de summapack.summa. Hay que hacerlo cuando cambian `sp-bootstrap.in`, `bajar-paquetes`, el diccionario o `fuentesexternas`; los paquetes no, esos van con `sp subir NOMBRE`. Hasta el 20/10/2026 era `make subir-bootstrap`, del Makefile a mano de summapack, que ya no existe.

`sp subir --comunes-fuentes summapack` sube `distros/salida/comunes/fuentes/` (las fuentes externas que baja `distros/espejo-fuentes`) a `comunes/fuentes/` del servidor (antes `make subir-comunes-fuentes`).

Ñ.10. subir-paquetes

Cada vez que hacer-pkg arma un proyecto, actualiza `subir.lista` en la raiz del repositorio (`siru/subir.lista`, `calcula/subir.lista`): lo ultimo que se armo de cada proyecto, con rutas completas. Para subir todo:

```
subir-paquetes $(cat ~/svn/siru/subir.lista)
subir-paquetes $(cat ~/svn/calcula/subir.lista)

subir-paquetes --prueba ARCHIVOS...   muestra lo que haria
subir-paquetes ARCHIVOS...            cada uno con su .md5 a PROYECTO/otros/
                                      y PAQ_UNSTABLE... apunta a el
subir-paquetes --estable PROYECTO...  lo UNSTABLE pasa a PROYECTO/, el
                                      estable anterior vuelve a otros/,
                                      el .summa se copia arriba (con
                                      VERSION=...) y el
```

```
                                    UNSTABLE desaparece hasta que se
                                    suba otra version
subir-paquetes --lista [PROYECTO...]  lo que hay en el servidor
subir-paquetes --reorganizar        una vez: lo que no sea de esta
                                    organizacion va a viejos/
```

El proyecto de cada paquete sale del .summa: el NOMBRE_VERSION.summa que deja hacer-pkg (conviene subirlo junto) o el del directorio del archivo. Lo que ya esta con el mismo md5 no se vuelve a subir; si esta distinto (una subida cortada), si.

El servidor: SUBIR_SERVIDOR, SUBIR_USUARIO, SUBIR_DIR, en `~/.config/simusol/subir`; lo que venga en el entorno manda. En simusol.org, www-data no entra por ssh: se usa el usuario publicador (del grupo www-data). Con `SUBIR_SERVIDOR=local` usa un directorio de esta maquina, para probar. `sp subir --bootstrap summapack` sube adonde dice el `SUBIR_A` de summapack.summa, como `sp subir`.

Ñ.11. bajar-paquetes e instalar-tgz

```
bajar-paquetes calcula                primero calcula.summa; muestra lo
                                      que hay (fuentes, deb, fedora, suse, arch, rpm,
                                      tgz...) y pregunta cual
bajar-paquetes --para deb calcula     los .deb, sin preguntar
bajar-paquetes --para fedora tabular2  los .rpm compilados en Fedora
bajar-paquetes --para arch tabular2/tabular   uno solo (y sus dependencias)
bajar-paquetes calcula.summa          los binarios que dice un .summa
                                      que ya se tiene
bajar-paquetes --instalar --para suse summapack  y los instala
bajar-paquetes --para tgz --inestable calcula
instalar-tgz calcula_6.0-38_amd64.tgz  instala con registro
instalar-tgz --lista | --archivos N | --quitar N
```

Ademas de lo pedido, baja las dependencias que son de nuestros proyectos y no estan instaladas (o estan en una version menor que la pedida): pedir tabular2/tabular trae tambien libtabular2-0, libslex0 y libunidades0. Las dependencias de cada paquete salen del .deb o .rpm bajado, y a que proyecto pertenecen, de los .summa del servidor. Las del sistema (libc6, python3) las resuelve apt o dnf. `--sin-dependencias` baja solo lo pedido.

bajar-paquetes comprueba el md5 de cada archivo y usa https con certificado (BAJAR_URL, por omision `https://simusol.org/downloads`). instalar-tgz es para las maquinas sin dpkg ni rpm: lleva registro en /var/lib/summapack/tgz para poder quitar lo que instalo, y no pisa archivos de otros paquetes.

Para instalar sin summapack (a mano, con curl y md5sum): doc/instalar-sin-summapack.md.

Para instalar summapack en una maquina que no lo tiene:

```
curl -fsSLO https://www.simusol.org/downloads/sp
sh sp --para deb calcula
```

(`sp subir --bootstrap summapack` sube a la raiz la guia, el sp autonomo --con bajar-paquetes adentro--, el diccionariodepaquetes y fuentesexternas. bajar-paquetes es autonomo: no averigua la distribucion (se la dice `--para`, o elige del menu), y con .tgz desempaqueta primero el de summapack, que trae instalar-tgz, y con el instala todo.)

Como sabe de que depende cada paquete (corregido el 07/10/2026, probando calcula en Fedora 44, openSUSE Tumbleweed y Arch peladas):

- .deb: su Depends; .rpm: lo que pide (`rpm -qpR`), y las bibliotecas (`libslex.so.0()(64bit)`) se pasan al nombre del paquete: `libslex0` (libX.so.N da libXN, o libX-N si X termina en cifra; `_` pasa a `-`).

- .tgz: el `var/lib/summapack/tgz/PAQUETE.control` que lleva adentro, que se lee solo con tar (una maquina pelada no tiene con que abrir un .deb). Los de Arch se compilan sin dh_shlibdeps: `distros/dentro.sh` agrega a ese Depends las bibliotecas que piden los binarios (NEEDED), con el mismo paso de nombre.
- Lo que es nuestro (esta en algun PROYECTO.summa del servidor) se baja e instala; lo demas es del sistema: con deb y rpm lo resuelve apt, dnf o zypper; con tgz se traduce con el diccionariodepaquetes (primero el del servidor, que es el mas nuevo) y se instala con pacman, dnf, zypper o apk.
- Los .md5: si un paquete se recompila con la misma version, hacer-pkg, dentro.sh y subir-paquetes rehacen su .md5; con uno viejo bajar-paquetes descarta el paquete ("bajo con otro md5").
- bajar-paquetes no usa awk (openSUSE pelado no lo trae).

Si a hacer-pkg le falta algo para compilar, lo dice antes y da la orden para instalarlo; `make dependencias` lo instala con packinstall.

Ñ.12. Donde estan las herramientas

Son parte de summapack, una sola copia:

archivo	en el arbol	instalado
`hacer-pkg.inc`	share/summapack/data/	/usr/share/summapack/data/
`summa.sh`	share/summapack/data/	/usr/share/summapack/data/
`diccionariodepaquetes`	share/summapack/data/	/usr/share/summapack/data/
`subir-paquetes`, `bajar-paquetes`, `instalar-tgz`	bin/	/usr/bin/

Los hacer-pkg de siru toman `hacer-pkg.inc` de `../summapack/share/summapack/data`, en el mismo repositorio. Los de calcula (otro repositorio) lo buscan en `SUMMAPACK_DATA` si esta puesta, despues en la copia de trabajo de siru al lado de la de calcula (`../siru`), y despues en summapack instalado. `summa.sh` se corre desde ahi mismo:

```
bash /usr/share/summapack/data/summa.sh debian DIR
bash ~/svn/siru/summapack/share/summapack/data/summa.sh debian DIR
```

Ñ.13. Proyectos con .summa (07/10/2026)

proyecto	paquetes
siru/summapack	summapack, libmem-util0, libmem-util-dev
siru/exprtk	libexprtk-dev
siru/unidades	libunidades0, libunidades-dev
siru/slex	libslex0, libslex-dev, d2datdinc
siru/tabular2	libtabular2-0, libtabular2-dev, tabular
siru/siru	siru
siru/lanzador	lanzador (05/11/2026; antes nuevosimusolsceptre)
calcula/calcula-6.0	calcula
calcula/puyuspa-6.0	puyuspa

Los nombres `_last` de antes (calcula_last_amd64.deb) quedaron en viejos/ y ya no se usan: la version estable la dice PROYECTO.summa.

Ñ.14. Publicar los documentos

Desde la raíz de summapack:

```
./subir-doc --prueba
./subir-doc
```

El paquete instala el comando en `/usr/bin/subir-doc`. Desde otra ubicación se puede ejecutar `subir-doc --doc ~/svn/siru/summapack/doc`; desde la raíz del proyecto alcanza con `subir-doc`. Los PDF no van incluidos en el paquete: el comando publica los documentos locales indicados. El paquete depende de rsync y openssh-client.

Archivo local	Dirección pública
doc/summapack.pdf	/downloads/summapack/docs/summapack-2026.pdf
doc/738.pdf	/downloads/summapack/docs/sumapack-jsl-2011.pdf

El script usa `SUBIR_SERVIDOR`, `SUBIR_USUARIO` y `SUBIR_DIR` de `~/.config/simusol/subir`; lo fijado en el entorno tiene prioridad. Requiere Bash, rsync y SSH. `--prueba` no conecta ni escribe. `SUBIR_SERVIDOR=local` permite probar contra un directorio local indicado por `SUBIR_DIR`.

Publica solamente esos dos PDF: no modifica paquetes, versiones estables ni el contenido de CMSimple. Los archivos sin cambios no se vuelven a transferir; los reemplazados se conservan en `docs/.anteriores/FECHA-ID/`. No elimina archivos ajenos. Si el .tex es más nuevo que el PDF, se detiene y pide regenerar el manual. Las subidas se preparan con nombres temporales mediante rsync antes de reemplazar el documento visible.

El directorio del proyecto `downloads/summapack/` debe existir antes de publicar. El script solo crea `docs/` dentro de él; si falta el proyecto, se detiene sin crear su directorio ni subir archivos. La comprobación se hace también en modo local. `--prueba` mantiene su carácter de simulación sin conexión: no verifica la existencia del directorio remoto.

En TinyMCE, seleccionar el texto del enlace, pulsar «Insertar enlace» y pegar la dirección relativa de la tabla. Funciona en los tres dominios del sitio. Las páginas HTML preparadas en `doc/web/` ya usan esas direcciones.

Prueba local del publicador: `python3 pruebas/prueba-subir-doc.py`.

bajar-paquetes no averigua el sistema (pedido del autor, 07/10/2026): lo que baja lo dice `--para`, y sin `--para`, en una terminal, lista con numeros lo que hay en el servidor para ese proyecto y se elige.

Ñ.15. Desde las fuentes: construir localmente, instalar el paquete

```
bajar-paquetes --para fuentes --instalar tabular2
```

El recorrido acordado tiene dos contextos distintos:

operación	entorno de construcción	resultado e instalación
Publicar binarios en el servidor	contenedor/chroot de la distribución elegida	paquetes destinados a descarga; construir no los instala en la máquina de desarrollo

operación	entorno de construcción	resultado e instalación
Instalar desde fuentes en una máquina	la propia máquina y sus bibliotecas de desarrollo	paquete construido localmente, instalado y desinstalable mediante el gestor local
Instalar manualmente desde un árbol de fuentes	la propia máquina	`make install` y `make uninstall` con el mismo prefijo; no queda registrado como paquete nativo

El instalador baja y comprueba los tarballs de fuentes, construye antes las dependencias de Simusol desde sus respectivas fuentes y pide al gestor local las dependencias externas. En Debian/Ubuntu usa `dpkg-buildpackage` y `apt-get install` sobre los `.deb` recién construidos. En Fedora/openSUSE usa la misma receta de construcción y `rpm_de_deb` con dependencias de bibliotecas detectadas en esa máquina, seguido de `dnf install` o `zypper install` sobre los `.rpm` recién construidos. El gestor correspondiente también los desinstala. `--prueba` anuncia el recorrido, pero no construye.

Verificación aislada del 12/10/2026: ExprTk se construyó desde un tarball y su paquete nativo se instaló, comprobó y quitó dentro de Debian 13, Ubuntu 24.04, Fedora 44 y openSUSE Tumbleweed. `make rpm` sobre una copia temporal de ExprTk produjo DEB, RPM, TGZ y fuentes; el DEB se instaló y quitó con APT en Debian 13 y el RPM con DNF en Fedora 44. SummaPack y `libmem_util` se construyeron e instalaron desde fuentes como RPM en Fedora 44 y openSUSE; la construcción DEB pasó en Debian 13. Unidades (biblioteca C) se construyó desde fuentes en Debian 13, Ubuntu 24.04, Fedora 44 y openSUSE; sus dos paquetes se instalaron, comprobaron y quitaron con APT, DNF y Zypper. Las imágenes Debian y Ubuntu ahora conservan el índice APT para poder instalar `bison` y otras dependencias nuevas durante las pruebas. Para los RPM se omite `dh_shlibdeps` del DEB intermedio porque no conoce las bibliotecas del sistema RPM; `rpmbuild` calcula esas dependencias sobre el binario local. La cadena completa de Simplot ya se probó en las cuatro distribuciones; faltan otros proyectos y publicar/verificar un bootstrap nuevo. No afirmar que la orden pública de Descargas tenga este cambio hasta hacerlo.

En Arch, el TGZ propio tiene `instalar-tgz --quitar`, pero no es un paquete de `pacman`: hace falta definir y probar el empaquetado nativo antes de afirmar que cumple el mismo contrato. La implementación actual se detiene allí sin instalación directa encubierta.

Prueba adicional en Fedora 44: `bajar-paquetes --para fuentes --instalar --inestable slex` resolvió y construyó en orden SummaPack con `libmem_util`, Unidades y Slex desde tres tarballs locales; DNF instaló los RPM resultantes, `d2datdinc` quedó ejecutable y la consulta RPM confirmó los seis paquetes requeridos. La desinstalación con DNF terminó correctamente. Fue una prueba en un contenedor efímero, no una publicación de esos artefactos.

Prueba de la cadena completa de Simplot en Fedora 44: `bajar-paquetes --para fuentes --instalar --inestable simplot` tomó tarballs locales de SummaPack, Unidades, Slex, ExprTk, Tabular2 y Simplot; construyó e instaló RPM nativos en ese orden mediante DNF. Pasaron las construcciones y las pruebas de empaquetado, `rpm -q` confirmó `simplot`, `tabular`, `libslex0` y `libunidades0`, y `command -v simplot` encontró `/usr/sbin/simplot`. Se corrigieron durante la prueba un prototipo interno faltante de Tabular2 y las equivalencias Fedora de dos módulos Perl y `gnuplot-nox`. El contenedor se descartó al terminar; no se publicaron paquetes ni se instaló nada en el anfitrión.

La misma cadena completa pasó después en Debian 13 con los mismos seis tarballs: APT instaló los DEB locales y `dpkg-query` confirmó `simplot`, `tabular`, `d2datdinc` y `libunidades0`; `simplot` quedó en `/usr/bin/simplot`. Los contenedores se descartaron al terminar.

El 14/10/2026 se repitió la cadena completa con tarballs locales actualizados de SummaPack 2017.2.1.R1008-16, Unidades 1.0-4, Slex 1.0-31, ExprTk 2025.01.01-2, Tabular2 2.0-20 y Simplot

2017.0.2.R1017-7. En Ubuntu 24.04, `bajar-paquetes --para fuentes --instalar simplot` construyó los seis proyectos y APT instaló los DEB locales; en openSUSE Tumbleweed, la misma orden construyó los RPM locales y Zypper instaló Simplot. Ambas órdenes terminaron con estado 0 dentro de contenedores efímeros, sin instalar nada en el anfitrión. No se comprobó en esta corrida la desinstalación ni el arranque de `simplot`; siguen pendientes la prueba del bootstrap público y las cadenas completas de otros proyectos.

Ñ.16. Catálogo histórico de fuentes externas (etapa 2, 07/10/2026)

La etapa histórica buscaba lo que pide BUILDDEPENDS y no es nuestro en `fuentesexternas` (share/summapack/data, y en la raiz del servidor): de donde sale el codigo original, su version y sha256, como saber si ya esta y como se compila. Los datos salen de Linux From Scratch y BLFS (no atados a ninguna distribucion); cada archivo se comprobo con el md5 de LFS/BLFS (ncurses, con la firma de su autor) antes de anotar la sha256. Si falta, se baja, se comprueba, se compila y se instala, con sus propias dependencias antes (autoconf necesita m4...). Las herramientas de base (gcc, make, perl, python3) no se compilan: se avisa si faltan.

Ese catálogo tiene lo que necesitaba la cadena de Tabular: m4, autoconf, automake, libtool, pkgconf, bison, ncurses, readline, pcre2. Probado en un Ubuntu 24.04 con solo compilador, curl y perl: con `bajar-paquetes --para fuentes --instalar tabular2` compila todo eso y nuestra cadena (10 minutos) e interpola andaba con el recorrido de entonces.

El catálogo incluye además CMake, GSL, OpenBLAS y SuiteSparse. En el recorrido automático vigente, las dependencias externas se piden al gestor de la distribución; `fuentesexternas` no se compila. Que una receta histórica exista no demuestra que toda la cadena de Calcula se haya probado en cualquier distribución.

Ñ.17. Que las fuentes de cada proyecto se preparen solas (etapa 3)

Cada tar.gz de fuentes trae `preparar` (hacer-pkg lo pone; la copia unica esta en share/summapack/data/preparar). Con las fuentes de cualquier proyecto:

```
./preparar
```

preparar lee el .summa del proyecto (viene en las fuentes) y, con bajar-paquetes --para fuentes --instalar --solo-dependencias, compila e instala paquetes construidos localmente de los proyectos Simusol requeridos, y las dependencias externas mediante el gestor de la distribución. Es una preparación de dependencias, no la instalación del proyecto actual. El `make install` del proyecto queda como alternativa manual; no sustituye la construcción del paquete. `--prefijo` no se admite en el recorrido automático de paquetes.

En los proyectos con makefile.lib: `make dependencias` hace lo mismo (y `make dependencias-distro`, con los paquetes de la distribucion via packinstall).

Ñ.18. Un solo juego de paquetes: base Ubuntu 22.04 y "comunes" (08/10/2026)

Compilar en cada distribucion (arriba, "Otras distribuciones") obliga a recompilar cada seis meses, y lo compilado en una no corre en otra porque cambian las bibliotecas (gsl 2.7 en Ubuntu 24.04, 2.8 en Debian 13 y Fedora). Lo que pidio el autor: **usar una distribucion vieja, compilar ahi todo lo nuestro y sus bibliotecas, y con esos ejecutables armar deb, rpm y tgz**. Cuando eso falle o se quiera algo mas moderno, se compila desde las fuentes (`--para fuentes`).

Por que anda:

- Lo compilado contra una glibc corre con las glibc posteriores (desde 1999, glibc 2.1), no al reves. La base es **Ubuntu 22.04** (glibc 2.35, GCC 11): corre en Ubuntu 22.04 y 24.04,

Debian 12 y 13, Fedora, openSUSE y Arch actuales. Debian 12 (glibc 2.36) dejaba afuera Ubuntu 22.04. Lo mismo vale para libstdc++ y libgfortran: se usan las del sistema.

- glibc y libm no se llevan: van pegadas al cargador del sistema (`ld-linux`), y llevarlas es lo que hacen AppImage o Flatpak, mucho mas fragil. Las funciones de libm pueden diferir en la ultima cifra entre versiones, nada mas.
- Las bibliotecas que si cambian de numero (gsl, KLU, OpenBLAS...) van con nosotros, siempre en la misma version, en **`/usr/lib/simusol`** (no chocan con las de la distribucion: pueden estar las dos). Nuestros binarios las buscan ahi (RUNPATH, puesto al compilar).
- Nada se optimiza para el procesador que compila: OpenBLAS va con `DYNAMIC_ARCH` (elige al correr segun el procesador).

Las piezas (en `distros/`):

`ubuntu2204/Containerfile`	el entorno Ubuntu 22.04
`comunes.summa`	el proyecto "comunes": PACKAGES (los cinco simusol-*), VERSION
`armar-bibliotecas`, `bibliotecas.sh`	compilan ncurses, readline, pcre2, gsl, OpenBLAS y SuiteSparse (KLU y Mongoose) de sus fuentes (las versiones, direcciones y sha256 de `fuentesexternas`) en /usr/lib/simusol, y arman simusol-bibliotecas (.deb, .tgz, .rpm), con un `shlibs` para que dpkg-shlibdeps ponga `Depends: simusol-bibliotecas` en lo nuestro
`compilar ubuntu2204`	compila nuestros proyectos ahi (dentro.sh): instala los simusol-* que hacen falta, no pide a la distri- bucion las -dev de esas bibliotecas, compila con `PKG_CONFIG_PATH=/usr/lib/simusol/lib/pkgconfig` y `-Wl,-rpath,/usr/lib/simusol/lib`, y de cada .deb arma el .tgz y el .rpm sin marca
`espejo-fuentes`	baja las fuentes de todo `fuentesexternas` (13, unos 160 MB) a `salida/comunes/fuentes/` con SHA256SUMS; `sp subir --comunes-fuentes summapack` las sube a `comunes/fuentes/`

En el servidor, `comunes/` (ver "El servidor de descargas"): los paquetes de los simusol-* y `fuentes/`, la copia de las fuentes de afuera. bajar-paquetes `--para fuentes` baja cada una de su direccion original y, si falla (o vino con otra sha256), de `comunes/fuentes/`; el campo `archivo:` de fuentesexternas da el nombre ahi cuando el de la direccion no sirve (SuiteSparse: `v7.14.1.tar.gz`).

Los paquetes de comunes, uno por biblioteca (pedido del autor, 08/10/2026: "como un paquete fuente con muchos binarios"), para que cada programa baje solo lo que usa (Tabular: readline y pcre2, no OpenBLAS):

paquete	trae	instalado
simusol-readline	readline 8.3 y ncurses 6.5 (usa la base de terminales del sistema)	1,7 MB
simusol-pcre2	pcre2 10.45	2,3 MB
simusol-gsl	gsl 2.8	5,3 MB
simusol-openblas	OpenBLAS 0.3.34 (BLAS y LAPACK, para todos los procesadores)	45 MB
simusol-suitesparse	SuiteSparse 7.14.1: KLU, Mongoose, AMD, COLAMD, BTF...	1,4 MB

Cada uno trae sus encabezados, .pc y archivos de cmake (no hay -dev aparte), y un `shlibs`: dpkg-shlibdeps pone en nuestros paquetes solo los que usan. Si se agrega un archivo que no corresponde a ninguno, bibliotecas.sh para y lo dice (funcion grupo()).

Probado (08/10/2026, servidor de prueba): el mismo calcula y el mismo Tabular se instalan y dan los mismos resultados en Ubuntu 22.04 y 24.04, Debian 12 y 13 (deb), Fedora 44 y openSUSE Tumbleweed (rpm) y Arch (tgz).

Con esto los paquetes `fedora`, `suse` y `arch` dejan de hacer falta: un .deb, un .rpm y un .tgz por paquete, iguales por dentro, para todas. `distros/fedora`, `suse` y `arch` quedan para probar.

Ñ.19. Producción incremental de todos los paquetes

Desde una copia SVN limpia, `./actualizar-paquetes --prueba` muestra qué proyectos requieren revisión sin construir ni publicar; `./actualizar-paquetes` recorre SummaPack, Unidades, Slex, ExprTk, Tabular2, Simplot, Dibuja, Calcula, Siru y Puyuspa en orden. También acepta nombres de proyectos para limitar la corrida, sin alterar ese orden. Compara la revisión SVN y la versión de cada proyecto con `distros/salida/estado-publicado.tsv`, que es un registro local no versionado y se actualiza únicamente después de una estabilización exitosa. Si no hay registro previo, hace una comprobación conservadora: reconstruye la fuente y la compara con la fuente estable del servidor.

Para cada proyecto que cambió, ejecuta su `hacer-pkg`, comprueba que la versión de `.summa` coincide con `debian/changelog`, construye en el contenedor Ubuntu 22.04 los DEB, RPM y TGZ de todos sus paquetes, verifica que estén presentes, sube solo esos archivos y las fuentes mediante `subir-paquetes`, y finalmente llama a `subir-paquetes --estable`. Una segunda ejecución sin cambios no construye ni sube nada. Si la fuente cambió pero conserva el número de versión ya publicado, se detiene antes de subir: primero hay que incrementar la versión en `.summa` y `debian/changelog`, para no reemplazar silenciosamente un paquete que los gestores consideran idéntico. Un fallo de construcción tampoco inicia la subida de ese proyecto. El comando exige `flock`, `curl`, SVN, las herramientas Debian de empaquetado y Podman; usa el servidor configurado para `subir-paquetes` y `BAJAR_URL` para consultar lo ya publicado.

La planificación y la segunda corrida sin cambios se probaron con un catálogo local el 14/10/2026. La corrida completa con publicación real todavía no se ejecutó; antes de usarla contra el servidor hay que revisar su plan con `--prueba` y las versiones de los proyectos que hayan cambiado. `installsumma.sh` y `bajar-paquetes` son el recorrido inverso: el primero consulta condicionalmente si cambió el ayudante que conserva en caché; el segundo enumera cada paquete propio con su versión instalada y la publicada, revisa las dependencias antes de omitir un proyecto y reconstruye solo lo que falta o está atrasado. En Debian/Ubuntu, el modo fuentes hace `apt-get update` una vez por ejecución y compara cada dependencia externa instalada con la versión candidata; actualiza solo las atrasadas mediante `apt-get install --only-upgrade`.

En Fedora y openSUSE refresca los repositorios y pide una actualización limitada a cada dependencia; no ejecuta una actualización general del sistema. En Arch no hace `pacman -Sy` aislado, porque dejaría una actualización parcial.

Ñ.20. Requisito: lo instalado desde fuentes se puede desinstalar (18/10/2026)

Pedido del autor: **cada paquete que `sp` instale desde fuentes deja su lista de instalacion, y se puede desinstalar con una orden de `sp`, sin las fuentes ni la carpeta donde se compilo.**

- Dos formas de instalar desde fuentes (decision del autor, 18/10/2026): **directa** (`make install` con su lista; la preferida, la del Manifiesto Anti-distros) y **como paquete** (con las fuentes se arma un .deb/.rpm y se instala con apt/dnf/zypper; util, se conserva). La directa anda en cualquier distribucion, Arch incluida.
- Donde: `PREFIX/var/lib/sp/instalados/NOMBRE.json` (la carpeta de estado de cada prefijo, como `localstatedir` de autotools, con `lib/` como pide la FHS, igual que `/var/lib/dpkg`): con `PREFIX=/usr/local`, `/usr/local/var/lib/sp/instalados/calcula.json`; con `/usr`, `/var/lib/sp/...`; en un prefijo personal, `~/.local/var/lib/sp/...`, sin ser root. `share/summapack/` queda para los archivos propios del paquete summapack; `var/lib/sp/` es lo que sp lleva de los paquetes que maneja, summapack incluido (pedido del autor, 18/10/2026: no en share, que es para datos que no cambian). Formato: el de `manifiesto-instalacion` (cada archivo con su sha256 y los directorios), mas el nombre y la version.
- Como se arma la lista, sin carpeta aparte (`make install` directo):
  - autotools (nuestros proyectos, gsl, pcre2, readline...): `make install INSTALL="summapack-instal -c"`; el envoltorio anota cada destino y llama al `install` verdadero. Los enlaces que libtool hace con `ln -s` (`libX.so`, `libX.so.N`) se agregan al final: los de la misma carpeta que apuntan a una biblioteca anotada.
  - cmake (SuiteSparse): se copia su `install_manifest.txt`.
  - Perl (simplot): se copia su `.packlist`, mas las paginas de manual.
  - summapack: su `install-manifest.json` pasa a esa misma carpeta (`var/lib/sp/instalados/summa`
- Desinstalar: borra solo los archivos intactos (misma huella); si alguno cambio o pertenece a un paquete de dpkg/rpm, no borra nada y lo dice.
- Comprobacion por proyecto (como `make distcheck`): instalar en un prefijo vacio, comparar lo que aparecio con la lista, desinstalar y ver que no quede nada.

Ñ.21. Prueba en los diez entornos (19/10/2026)

`sh sp --para fuentes siru puyuspa` (directa: make install con su lista, y `sp remove` de cada proyecto) y `sh sp --para fuentes-paquete siru puyuspa` (.deb/.rpm armados ahi, instalados y quitados con apt/dnf/zypper), en las imagenes `simusol-*:desarrollo`: toda la cadena (summapack, unidades, exprtk, slex, tabular2, simplot, calcula, siru, puyuspa), una cuenta de calcula, y despues de desinstalar no queda nada nuestro.

entorno	directa	como paquete
Ubuntu 22.04 y 26.04	anda	anda
Debian 12 y 13	anda	anda
Fedora 36 y 44	anda	anda
openSUSE Tumbleweed	anda	anda

entorno	directa	como paquete
openSUSE Leap 15.4	anda	no: su debhelper 13 no carga con su Perl 5.26; sp lo dice y pide --para fuentes
Arch 2022 y actual	anda	(no tiene esta forma)

Lo que hubo que arreglar para llegar ahi: seguir tambien lo que cada proyecto pide para correr (DEPENDS: siru pide calcula); como paquete, lo del sistema para correr lo resuelve el gestor y sin Recommends (gnuplot-qt contra gnuplot-nox); en Fedora/openSUSE, reglas vacias para los pasos de debhelper que suponen Debian y base de dpkg vacia; Arch: core_perl en el PATH y pacman -Syu; diccionario con alternativas `a|b` (pkg-config en Leap); sp-instalados con Python 3.6; los *.tab.c del tar.gz mas nuevos que su .y (bison 3.0.4 de Leap); compilador basico (build-essential/base-devel) si falta.

Ñ.22. Los .rpm desde el .summa: summa.sh rpm (19/10/2026)

Pedido del autor: "que de summa pase a la info de los rpm". `summa.sh rpm DIR [--marca M] [--destino D] [--tar T]` escribe `NOMBRE.spec` desde el .summa, como `summa.sh debian` escribe `debian/`; asi en las distribuciones de rpm no hacen falta dpkg ni debhelper (el debhelper de Leap 15.4 no anda). Junta la plantilla vieja `template.spec` (compilar e instalar con make DESTDIR) y lo que resolvia `rpm_de_deb` de hacer-pkg.inc:

en el .summa	en el .spec
VERSION	Version y Release (+ .MARCA: fedora, suse)
SUMMARY, DESCRIPTION, LICENSE, HOMEPAGE, MAINTAINER	lo mismo
BUILDDEPENDS	BuildRequires (diccionario, columna RPM)
DEPENDS, NOMBRE_DEPENDS, Recommends/Provides/Conflicts de NOMBRE_EXTRA	Requires, Recommends...; las bibliotecas las calcula rpm
FUENTES_CONSTRUIR / FUENTES_INSTALAR	%build / %install (si no: %configure CONFIGURE, make install)
PACKAGES, ARCHIVO_NOMBRE_install	un subpaquete por paquete, con su %files

Detalles: DESTDIR va tambien por MAKEFLAGS (un Makefile que define `DESTDIR =` no toma el del entorno); el principal es noarch solo si todos los paquetes son "all"; las listas aceptan "ARCHIVO" y "ARCHIVO CARPETA" (como dh_install), `usr/lib/*/` se busca tambien en lib64 y lib, las paginas de manual e info van con `*` (rpm las comprime), los `.la` no van (Fedora los borra); con un solo paquete va todo lo instalado. LIBDIR y PREFIX van en el entorno de %build/ %install (summapack se los pasa a mem_util). `sp --para fuentes-paquete` en Fedora y openSUSE arma asi, con rpmbuild; en Debian/Ubuntu sigue el .deb. Probado: la cadena entera (16 paquetes) en Fedora 36 y 44, Tumbleweed y Leap 15.4.

Ñ.23. Los binarios de las bases viejas: armar-paquetes y probar-paquetes (19/10/2026)

Pedido del autor: los .rpm no en su maquina sino en el podman de cada distribucion, "uno para suse otro para fedora".

```
distros/armar-paquetes [ENTORNO...]    (por omision ubuntu-viejo fedora-viejo suse-
    viejo)
```

```
distros/probar-paquetes [ENTORNO...]   (por omision los 8 viejos y nuevos con paquetes
    )
```

armar-paquetes corre en `simusol-ENTORNO:desarrollo sp --para fuentes-paquete siru puyuspa` (toda la cadena) con `SP_GUARDAR_PAQUETES`, y deja los .deb/.rpm con su .md5 en `salida/viejas/FAMILIA/` (lo de un entorno actual, en `salida/actuales/`); de ahi los toma `sp subir`. Las fuentes de cada proyecto: las mas nuevas entre la copia de trabajo (lo que anoto hacer-pkg en subir.lista) y las publicadas. probar-paquetes instala eso en las maquinas viejas y nuevas de cada familia, prueba calcula y Tabular y desinstala. Ninguno borra nada: cada corrida usa carpetas nuevas (`salida/servidor-local-FECHA`, `salida/pruebas-FECHA`).

Resultado (19/10/2026): andan en la vieja donde se armaron (Ubuntu 22.04 y Debian 12, Fedora 36, Leap 15.4) y no en las nuevas (Ubuntu 26.04, Debian 13, Fedora 44, Tumbleweed): calcula pide la gsl y la KLU de la vieja (libgsl27, libklu1; libgsl.so.25) y en Tumbleweed calcula y Tabular piden libreadline.so.7. Desde fuentes (directa o como paquete) anda en todas. Pendiente: comunes en las bases viejas, o que sp pase solo a compilar cuando el paquete armado no se puede instalar.

Ñ.24. Desde el arbol de svn: --arbol (19/10/2026)

Pedido del autor: que la instalacion desde fuentes pueda usar el arbol en vez del servidor, y que suba la version cuando haga falta.

```
sp --para fuentes-paquete --arbol install tabular2
sp --para fuentes --arbol ~/otro install calcula
```

El usuario usa solo `sp` (pedido del autor, 19/10/2026): sus opciones `--para FORMA` (fuentes, fuentes-paquete, o un formato armado: deb, fedora, suse, arch) y `--arbol [DIR]` las pasa a bajar-paquetes por el packlet de summapack; bajar-paquetes queda como herramienta interna (y es lo que lleva adentro el `sp` de la raiz de descargas).

Con `--arbol [DIR]` (por omision `~/svn`), para cada proyecto de la cadena busca NOMBRE.summa en el arbol (la carpeta con hacer-pkg; si hay varias, la de VERSION mas alta) y compila esa carpeta tal como esta (`svn export` de la copia de trabajo: lo comiteado y lo que no), con FUENTES_PREPARAR. Lo que no esta en el arbol se baja del servidor. Sin `--arbol`, las fuentes son las publicadas (como para cualquier usuario).

La version (desde el 20/10/2026): la del arbol, AÑO.MAYOR.MENOR.rREV-N (sp-version; ver "Versiones"): si el arbol tiene commits que lo publicado no tiene, la revision ya es otra; si tiene cambios sin comitear, lleva +local (2026.6.0.r310+local-1). En los dos casos es mayor que la publicada y no se confunde con ella. No se toca el arbol: la version completa se escribe en la copia que se compila. (Antes subia el -N en el .summa y en debian/changelog del arbol y avisaba "comitealo".)

Sin svn (otro sistema de versiones: git, hg...; o ninguno; 20/10/2026): se copia lo que dice el MANIFIESTO y la version lleva +local.

Ñ.25. MANIFIESTO: las fuentes de cada proyecto (20/10/2026)

Pedido del autor: que las fuentes del paquete no dependan del sistema de versiones. Cada proyecto tiene un `MANIFIESTO` junto a su .summa: una ruta por renglon (relativa a la carpeta; `#` comenta), el mismo versionado.

- Manda el manifiesto: lo que esta ahi va en las fuentes (el tar.gz); lo que no, no va, aunque este en svn (respaldos, notas). No hace falta `svn add`: se agrega al manifiesto.
- El sistema de versiones se adapta: cualquier `sp` sobre un proyecto (`sp paquete`, `sp subir`, `sp install --arbol`) y `hacer-pkg` empiezan sincronizando: lo del manifiesto que no este versionado se agrega (svn add, git add, hg add), sin comitear, y lo avisa; si el manifiesto

nombra un archivo que no existe, error. Con `--arbol` (sp corre con sudo) se sincroniza como el dueno del arbol, para no dejar .svn de root.
- Sin sistema de versiones, el manifiesto solo alcanza.
- Las fuentes se copian con `sp-manifiesto copiar`: iguales con svn, git o nada.
- `sp manifiesto [LUGAR/]NOMBRE` lo genera si no esta (de svn: la copia local; de git ls-files; de hg files; o de la carpeta sin restos de compilacion) y si esta sincroniza. Los de la cadena (summapack, unidades, exprtk, slex, tabular2, simplot, siru, calcula, puyuspa 7.0) se generaron desde svn: hacer-pkg arma las mismas fuentes que antes (mas el MANIFIESTO).

El manifiesto dice que es fuente; que se INSTALA lo deciden make install y las listas del .summa (ver "Que decide el .summa").

Ñ.26. Versiones: AÑO.MAYOR.MENOR.rREV-N (20/10/2026)

Decision del autor: un solo esquema para todos, con el año, un numero principal, uno secundario y, si hay svn, la revision.

parte	ejemplo	quien la pone
AÑO.MAYOR.MENOR	2026.6.0	una persona, en el .summa (VERSION y BASEVERSION)
.rREV	.r310	`sp-version`: la ultima revision de svn que toco la carpeta del proyecto
+local	+local	`sp-version`: hay cambios sin comitear, o no hay svn
-N	-1	una persona, en el .summa: el empaquetado; sube solo si se rearma el mismo codigo

El .summa de svn dice `VERSION="2026.6.0-1"`; hacer-pkg (y `sp --arbol`) escriben en la copia que arman `2026.6.0.r310-1`, en el .summa y en la primera entrada de debian/changelog, y de ahi salen el tar.gz, los paquetes, AC_INIT (SP_VERSION) y el .pc. Asi cada commit da una version nueva sin tocar el .summa, lo publicado dice de que revision salio, y `sp subir` rearma solo si hubo un commit despues de armar (el nombre cambia). Las fuentes publicadas ya traen la version completa: `sp-version` no les agrega nada.

La revision es de cada repositorio (siru va por r4424, calcula por r310): dentro de un proyecto siempre crece; solo se romperia si el proyecto se mudara de repositorio. Todas las versiones nuevas son mayores que las viejas, para apt y para rpm (probado con dpkg --compare-versions y rpm.vercmp), asi que se actualizan solas.

proyecto	antes	ahora (con la revision del 20/10/2026)
summapack	2017.2.1.R1008-20	2026.2.1.r4424-1
unidades	1.0-4	2026.1.0.r4424-1
exprtk	2025.01.01-3	2025.01.01.r4424-1 (la fecha es la del autor original, Arash Partow)
slex	1.0-37	2026.1.0.r4424-1
tabular2	2.0-25	2026.2.0.r4424-1
calcula	6.0-43	2026.6.0.r310-1

proyecto	antes	ahora (con la revision del 20/10/2026)
simplot	2017.0.2.R1017-9	2026.0.2.r4424-1
siru	1.0-7	2026.1.0.r4424-1
puyuspa	7.0-4	2026.7.0.r310-1

Antes cada uno tenia el suyo (el de 2017 de makefile.lib con una revision congelada, mayor.menor clasico, la fecha de exprtk) y el -N contaba todos los cambios.

Ñ.27. Que decide el .summa (20/10/2026)

Regla del autor: "todo lo que deba decidirse al summa"; "lo que sea automatico no". Va en el .summa lo que alguien decide: nombre, version, descripcion, licencia, los paquetes y que archivo va en cual, adonde se instala cada cosa, las dependencias que no se pueden deducir (perl, gnuplot, calcula para siru), las opciones de configure que se eligen. No va lo que se calcula solo: las dependencias de bibliotecas (dpkg-shlibdeps, rpm), los archivos instalados (los anota make install), lib64/multiarch, la marca de la distribucion, los .md5, debian/ y el .spec (se generan). Proximo: `summa.sh automake` generara la parte de instalacion de Makefile.am (instalar.am) desde el .summa; la compilacion (fuentes, LDADD, condiciones) queda en Makefile.am.

Pedido del autor (20/10/2026): todos los paquetes con autotools, tambien los de perl, y "solo habra dos fuentes de info: los .summa y MANIFIESTO".

`summa.sh automake DIR` genera:

- `instalar.am`: que se instala y adonde, de las listas `NOMBRE.install` del .summa (las mismas que usan el .deb y el .rpm);
- `dist.am`: que va en el tar.gz, lo del MANIFIESTO;
- `configure.ac` y `Makefile.am`, si el proyecto no compila nada (no los trae en el MANIFIESTO). Los que compilan (C, C++, Fortran) escriben los suyos con lo de compilar y hacen `include instalar.am` e `include dist.am`.

Nada de eso va en svn (`svn:ignore`). hacer-pkg lo genera al armar el tar.gz publicado y ademas corre `autoreconf -fi` (20/10/2026, pedido del autor): el tar.gz trae `configure`, los `Makefile.in` y `ltmain.sh`, como cualquier paquete GNU. La maquina que instala desde ese tar.gz hace solo `./configure && make && make install` (lo que hace sp --para fuentes): no necesita autoconf, automake ni libtool, ni que sean de la misma version, y no estan en los BUILDDEPENDS (para el .deb los trae debhelper). Al desempaquetar se respetan las fechas del tar, todas iguales: con `tar --touch`, `configure.ac` podia quedar mas nuevo que `configure` y make queria rehacerlo. Probado el 20/10/2026: los nueve tar.gz compilan e instalan en el podman de Ubuntu actual sin autoconf, automake ni libtool.

Desde la copia de svn no hay configure: `sp --arbol` instala autoconf, automake y libtool y corre `autoreconf -fi` en la copia; a mano: `summa.sh automake . && autoreconf -fi && ./configure && make check`.

Los renglones de una lista de instalacion:

renglon	que hace
`ORIGEN DESTINO`	copia ORIGEN (archivo, carpeta entera o patron con `*`) a la carpeta DESTINO
`ORIGEN DESTINO NOMBRE`	lo mismo, con otro nombre
`RUTA`	lo instala la compilacion (Makefile.am a mano): solo dice en que paquete va

renglon	que hace
`!RUTA`	no se copia, aunque este adentro de una carpeta de la lista
`enlace RUTA VALOR`	un enlace simbolico; un enlace del MANIFIESTO se instala como enlace

DESTINO es la ruta del paquete (`usr/bin`, `usr/share/perl5`, `etc`, `var/www/...`); automake la traduce a `$(bindir)`, `$(datadir)/perl5`, `$(sysconfdir)`, `$(localstatedir)/www/...`, asi anda con cualquier `--prefix`. Ejecutable (755): lo que va a una carpeta bin, lo que termina en .pl, .sh, .py o .cgi y lo que ya lo es en las fuentes. Campo nuevo del .summa: `PRUEBAS`, las pruebas de `make check` (.t con el conductor TAP, .py, .sh); sin el campo, todas las `t/*.t` y `tests/*.t` del MANIFIESTO. Una carpeta del MANIFIESTO con su propio `configure.ac` (mem_util en summapack) es un subproyecto: `AC_CONFIG_SUBDIRS` y `SUBDIRS`. Los nombres de archivo con espacios u otros caracteres que make no acepta se copian uno por uno, pero conviene no usarlos (los `UTUTO-...` de summapack se renombraron).

proyecto	estado (20/10/2026)
simplot, siru, puyuspa, exprtk, summapack	configure.ac y Makefile.am generados del .summa y el MANIFIESTO; sin Makefile a mano
unidades, slex, tabular2, calcula	configure.ac y Makefile.am a mano solo con lo de compilar (y summapack.m4); instalar.am y dist.am generados

Probado en cada uno: make check, make distcheck y el .deb; los .deb salen iguales a los publicados (summapack: se agrega sp-manifiesto y dejan de ir a doc/ las copias de scripts que ponia makefile.lib).

Lo que repiten los configure.ac de los que compilan esta una vez, en `summapack/share/aclocal/summapack.m` (summapack lo instala en `share/aclocal`; `summa.sh automake` lo copia al `m4/` de cada proyecto, asi va en el tar.gz): `AC_INIT([NOMBRE], SP_VERSION, SP_CORREO)` (version y correo del .summa; el nombre va escrito porque autoreconf busca AC_INIT), `SP_BISON([3.6])`, `SP_PROPIA([VAR], [modulo], [paquete])` (una biblioteca nuestra por pkg-config, y que instalar si falta), `SP_READLINE`, `SP_ASPRINTF`.

Los objetivos propios del Makefile a mano de summapack pasaron a sp: `sp subir --bootstrap summapack` (antes `make subir-bootstrap`) y `sp subir --comunes-fuentes summapack`; el sp autonomo lo arma `distros/armar-sp`.

Ñ.28. Referencia, seguridad y límites

El manual `doc/summapack.tex` reúne la referencia de todos los componentes, con ejemplos y efectos. `BIN.md` sirve de índice de las herramientas actuales y conserva el relevamiento histórico; `mem_util/README` documenta la biblioteca C.

- `.summa`, `fuentesexternas`, plantillas y la configuración de publicación son material de confianza: pueden ejecutar órdenes. No deben tomarse de un origen desconocido.
- MD5 permite detectar diferencias respecto del dato publicado, pero no es una firma. Las fuentes externas usan SHA256 del catálogo; hay que confiar también en ese catálogo.
- `sh sp --para fuentes PROYECTO` instala directamente en `/usr/local` o en `--prefijo`, con registro de archivos. `sh sp --para fuentes-paquete PROYECTO` construye e instala paquetes nativos. La guía canónica (`https://www.simusol.org/downloads/`

`0-README-como-instalar`) es la referencia vigente; los resultados fechados de 2026 describen el recorrido disponible cuando se hicieron esas pruebas.

- Los once proyectos activos exponen `make pkg`, `make deb`, `make rpm` y `make tgz` como acceso a `./hacer-pkg`; estos objetivos se verificaron sintácticamente con `make -n pkg`. La generación efectiva de RPM requiere `rpmbuild`; la prueba de instalación de toda la cadena de proyectos sigue pendiente.
- La detección por `PKGCONFIG` comprueba presencia; no es un resolvedor completo de restricciones de versión. Las dependencias de base o fuera de catálogo se informan y pueden necesitar intervención.
- `findbinlib` sin BORRA informa, pero crea un directorio auxiliar en /tmp; con BORRA mueve archivos. `icon_install` puede tolerar fallos individuales del escritorio.
- `make prepare` es el objetivo histórico de páginas de manual; `./preparar` y `make dependencias` preparan dependencias desde fuentes. `make dependencias-distro` utiliza los paquetes de la distribución.

O. makefile.lib: catálogo

`makefile.lib` es la biblioteca de make de summapack (Diego Saravia, proyecto Simusol, desde 2016). Un proyecto la incluye y recibe hecho lo común: la versión, instalar con `DESTDIR` y `PREFIX` llevando la lista de lo instalado, armar tar.gz, .deb y .rpm, firmarlos, subirlos al servidor e instalar dependencias. Este catálogo dice qué trae, cómo se usa y qué anda hoy. Se escribió el 07/10/2026, leyendo el código y probándolo con `pruebas/prueba-lib.sh` (sección 6).

`make ayuda`, en cualquier proyecto que la incluya, lista los objetivos con una línea cada uno.

O.1. 1. Cómo se usa: el proyecto mínimo

`summapack/pruebas/ejemplo` es el proyecto más chico que la usa, y la referencia de este catálogo:

```
ejemplo/
  ejemplo.summa        la versión y la descripción (sección 2)
  Makefile             incluye la biblioteca y define install
  bin/hola.sh          un programa: se instala como bin/hola
  share/ejemplo/...    datos: van a PREFIX/share
  etc/ejemplo.conf     configuración: va a /etc
  README COPYING changelog INSTALL.txt   van a la documentación
```

El `Makefile`:

```
all:

MAKEFILE_LIB ?= /usr/local/share/summapack/data/makefile.lib
include $(MAKEFILE_LIB)

install:  ## instala el ejemplo en $(DESTDIR)$(PREFIX)
    $(call copytree,$(DESTDIR)$(PREFIX),$(DESTDIR),,$(PREFIX))
```

- `all:` va **antes** del `include`: así el objetivo por omisión es el del proyecto y no uno de la biblioteca.
- `copytree` instala todo según dónde está en el árbol (sección 4). Con eso ya andan `make install`, `dist`, `tgzbuild`, `debuild`, `ayuda`.
- Un comentario `## texto` en la línea de un objetivo lo hace aparecer en `make ayuda`.
- La biblioteca busca el `.summa` en `.`, `..` y `../..`, y usa el más cercano (desde el 07/10/2026; antes pegaba todos los que hubiera, y un proyecto adentro de otro quedaba sin versión).

Dónde está la biblioteca. Hay **una sola copia**, la de summapack (`summapack/share/summapack/data/ma`
y los proyectos usan la **instalada**: la buscan con `findfirstSB` en `/usr`, `/usr/local` y `~` (con `test -x`: tiene que ser ejecutable). Se instala sola, con su catálogo:

```
make -C summapack instalarlib PREFIX=$HOME        # para el usuario, sin root
sudo make -C summapack instalarlib PREFIX=/usr/local
```

Sin ella instalada, esos Makefiles se saltean el `include` sin avisar y no tienen los objetivos de la biblioteca.

Hasta el 07/10/2026 había cuatro copias en siru. `sit` y `simplot` llevaban la suya en `share/`, así que **al instalarlos pisaban la instalada**; `sit` además la usaba antes que la instalada (su `findfirstSB` mira primero `.`) y tenía dos ajustes propios (el usuario del servidor y `SERVERBAJAR` fijo) que ahora están en `sit/Makefile`. Las dos copias se sacaron. La de `tabular2/viejos/interpola` queda: es código guardado como referencia, que no se compila ni se instala (y es de 2018, aunque la fecha del directorio diga 2026).

Quiénes la usan. 19 Makefiles de siru la incluyen, más summapack, que usa su propia copia (la de referencia):

proyecto	último cambio	cómo
summapack	2026-10	su copia, la de referencia
exprtk	2026-10	la instalada
sceptresimusol, siru/corpus/sceptre-build	2026-10	una copia local o la instalada (`findfirstSBX`)
sit	2021-11	una copia local o la instalada
simusol-2017.0.1, simusol-2017.0.2-Nilsa, simusol-2017.0.3, simusol-ASUS, simusol-doc, simusol-examples	2021-08	la instalada
simusol-PD, simusol-SinPlot, simusol-PD_endesarrollo	2019	la instalada
Radia, Plot, Base, dia-simusol	2018	la instalada
make	2016	la de `/usr` o `/usr/local`

Con la biblioteca nueva instalada (en un home de prueba), `make ayuda` anda en exprtk, sceptresimusol, sceptre-build, sit y Base, y leerlos no deja archivos.

O.2. 2. El `.summa` y los archivos de configuración

`NOMBRE.summa` (el nombre del archivo es el del paquete):

campo	qué es
`BASEVERSION=1.0.0`	la versión; la biblioteca le agrega `.R<revisión de svn>`
`DESCRIPTION`, `SUMMARY`, `KEYWORDS`, `AUTHOR`, `MAINTAINER`, `DEPENDS`	para los paquetes (los usa `templatesumma` al armar el .spec y debian/)
`SIMUSOLTYPE`	el tipo de paquete (variable `TYPE`)

Archivos sueltos optativos, al lado del `.summa`: `TIPOPACK` (noarch, x86_64...), `TIPODEB` (all, amd64...), `LIBVERSION` (si es una biblioteca), `../FURY` (el token de Gemfury para `uploaddeb` y `uploadgem`).

`etc/summapack.yaml` (o `~/etc/`, o `/etc/`): `SERVERBAJAR`, `SERVERDIR`, `SERVERUSER` para subir al servidor. `SERVERNAME` está fijo en `simusol.org` aunque el yaml diga otra cosa (el propio archivo lo marca: "MAL GREP SERVERNAME").

O.3. 3. Variables

Las que se dan al llamar (`make VARIABLE=valor ...`):

variable	qué hace	por omisión
`DESTDIR`	raíz de la instalación (para armar paquetes)	vacía
`PREFIX`	dónde se instala	`/usr/local`
`ASUSER=1`	instalar para el usuario, en `$HOME`, sin sudo	
`SUMMA_CHARLA=1`	muestra los mensajes de la biblioteca (el paquete, la versión, la distribución, el servidor, los temporales...). Sin ella no dice nada al leerse	
`DEBUG=1`	muestra las listas de archivos que va a instalar	
`DEBIANREVISION`	revisión del .deb	1
`SERVERSUBIR`, `SERVERDIR`, `SERVERUSER`	adónde sube `upload`	del yaml, o simusol.org

Las que la biblioteca calcula y un proyecto puede usar:

variable	qué es
`PACK`, `PACKV`	el nombre del paquete (`PACKV` con `_` en lugar de `-`)
`VERSION`, `VERSIONSUMMA`, `SVN_REV`	`BASEVERSION.R<rev>`, la del `.summa`, la revisión
`DISTRO`	`UBUNTU`, `DEBIAN`, `SUSE`... (con el programa `distro` de summapack, o de `/etc/os-release` si no está)
`DOWNLOAD`	caché de lo bajado: `/var/cache/summapack` o `~/var/cache/summapack`
`PKGDIR`	la documentación: `PREFIX/share/doc/packages/PACKV`
`IBINFILES`, `ISHAREFILES`, `ILIBFILES`, `IETCFILES`, `ISRCFILES`, `IPKGFILES`	los archivos del proyecto por directorio (`IPKGFILES`: los de arriba)
`ISHAREDIRS`, `ILIBDIRS`, `ISRCDIRS`	los directorios
`BASETMP`, `BASETMPMAN`	el nombre de un directorio temporal en `/tmp`, sin crearlo: el objetivo que lo usa lo crea (`mkdir -p /tmp/$(BASETMP)`) y lo borra
`USERNAME`, `homedir`, `VEROID`	quién corre (con sudo, el usuario de verdad), su home, si es root
`WEBDOC`, `WEBCGI`, `USR`, `GRP`	rutas y usuario del servidor web según la distribución
`FC`	el compilador de Fortran (gfortran; g77 en Cygwin)

O.4. 4. Funciones (`$(call nombre,arg1,arg2,...)`)

O.4.1. Instalar

Todas anotan lo que instalan en `listado.ls` (`D:` directorio, `A:` archivo), que `listado` después guarda como registro del paquete, para poder desinstalarlo.

función	argumentos	qué hace
`copytree`	1 destino con prefijo (`$(DESTDIR)$(PREFIX)`), 2 `DESTDIR`, 3 `sudo` o vacío, 4 prefijo para el listado	**la que usa un proyecto**: instala `share/`, `lib/`, `bin/` (sin `.sh` ni `.pl`), los archivos de arriba en `PKGDIR` (el changelog comprimido; `COPYING` e `INSTALL.txt` como `Upstream-...`) y `etc/` en `DESTDIR/etc`
`installdirs`	1 directorios, 2 destino, 3 prefijo a quitar, 4 sudo, 5 ruta para el listado	crea los directorios
`installfiles`	1 archivos, 2 destino, 3 prefijo a quitar, 4 sudo, 5 origen alternativo, 6 ruta para el listado	instala archivos
`installbinfiles`	igual	instala programas sacándoles `.sh` o `.pl`
`installetcfiles`	1 archivos, 2 `DESTDIR`, ...	instala `etc/` en `DESTDIR/etc`, con sus subdirectorios, sin permiso de ejecución
`installsrcdirs`, `installsrcfiles`	como las de arriba	lo mismo con `src/`, solo si `SRCDIR=src`
`listado`	1 raíz	agrega `listado.ls` a `PREFIX/share/PACK/ls/PACK.ls` y lo borra

O.4.2. Bajar, buscar, quitar

función	argumentos	qué hace
`download`	1 dónde está ya instalado (vacío: no está), 2 caché, 3 nombre del archivo, 4 URL, 5 raíz del directorio del tar, 6 nombre para los mensajes, 7 comando de instalación	baja si el md5 del servidor cambió, verifica el md5, desempaqueta en `BASETMP` y corre el comando. Prueba `UNSTABLE` y si no, `last`; desde el 07/10/2026 esos nombres solo estan en downloads/viejos/: para bajar, `bajar-paquetes` (ver PAQUETES.md)
`mkmd5`	1 directorio, 2 archivo, 3 URL	baja el archivo, calcula su md5 y **lo sube al servidor**

función	argumentos	qué hace
`findfirst`	1 ruta (`bin/x`)	la ruta completa del primer ejecutable en `/usr`, `/usr/local`, `~`
`findfirstSB`	1 ruta	el prefijo (`/usr`, `/usr/local` o `~`) donde está
`findfirstS`	1 ruta	el primer archivo no vacío en el proyecto, `~` o `/`
`findfirstSBX`	1 ruta	el prefijo: `.`, `~`, `/usr/local` o `/usr`, el primero que lo tenga
`rmbin`, `rmfbin`	1 texto, 2 nombre	borran un programa de `/usr/local/bin`, `/usr/bin` y `~/bin` (con sudo si se puede)
`compilation`		`make` y `sudo -n make install`
`sedmake`, `sedmake2`		reemplazos en `simusol-make` [viejo]
`SIMUSOLBASH`		el texto de `~/.simusolbash` (arma el PATH); lo usa `changebash`

O.5. 5. Objetivos

`make ayuda` los lista. El estado sale de `pruebas/prueba-lib.sh` el 07/10/2026 (Ubuntu 24.04):

objetivo	qué hace	estado
`ayuda`	lista los objetivos	**anda**
`install` (del proyecto, con `copytree`)	instala	**anda**
`dist`	tar.gz de las fuentes con la versión, al lado del proyecto	**anda**, en svn y fuera de él
`tgzbuild`	`-AI.tgz` instalable, con `make install` en un temporal	**anda**
`clean`, `deepclean`	borran `*~` y `listado.ls`; las páginas de manual generadas	**anda**
`prepare`, `plshman`	páginas de manual de `bin/` con help2man	sin probar (falta help2man)
`debuild`	.deb, con debian/ de `templatesumma`	sin probar (falta `templatesumma` instalado). Para un paquete Debian de verdad, `hacer-pkg.inc` (sección 8)
`rpmbuild`	rpm y src.rpm firmados	sin probar (falta rpmbuild); tiene `/home/dsa/rpmbuild` fijo

objetivo	qué hace	estado
`upload`, `stable`	suben el tar.gz al servidor, lo pasan a `last`	no se prueba: **publica**. Desde el 07/10/2026 el servidor esta organizado por proyecto y sin `last` (lo anterior quedo en downloads/viejos/): para publicar, `subir-paquetes` (ver PAQUETES.md)
`uploaddeb`, `uploadgem`, `rpmbuildosc`, `bajaroscrpm`	Gemfury y el Open Build Service de openSUSE	no se prueba: **publica**. Las dos de Gemfury leen el token de `../FURY`. `rpmbuildosc` usa `~/osc/tabular` y `~/svn/siru` fijos
`gpgsign`, `changebash`, `idesktop`, `installsummapackMAKE`	clave gpg; `.simusolbash` en cada usuario; lanzadores; instala summapack	no se prueba: **cambia la máquina**
`cpmakefilelib`	copia la biblioteca de `../summapack`	trivial
`exprtkhpp`, `exprtkbajar`	bajan exprtk	**viejo**: hoy está el paquete `libexprtk-dev`
`solosolopack`, `inslaunchersXXX`	tar sin .svn; lanzadores	**viejo**
`cygwin`, `diawin`, `cyggfortran`, `gnuplotW`, `emacswin`, `strawberry`	instaladores de Windows	**viejo** (2012-2016)
`unzip libgsl-dev gsl-bin ... check`	instala con apt el paquete que se nombre: `make libgsl-dev`	sin probar; nombres de paquetes de 2017 (`libgsl2`)
`File/Slurp Cwd ...`	instala con cpan el módulo de Perl que se nombre	sin probar

O.6. 6. Cómo se prueba

```
summapack/pruebas/prueba-lib.sh [MAKEFILE.LIB]
```

Copia el ejemplo a un temporal, corre cada objetivo que se puede correr sin publicar ni cambiar la máquina, y comprueba lo que dejó. **OK** anda, **FALLA** no anda (la prueba sale con 1), **PENDIENTE** es un problema conocido de la sección 7: avisa sin hacer fallar, y cuando se arregla pasa a OK sola. Después de los arreglos del 07/10/2026 (sección 7): 29 OK, 0 FALLA, 0 PENDIENTE; también se probó `dist` y `tgzbuild` de summapack mismo.

O.7. 7. Problemas: los arreglados y los que quedan

Arreglados el 07/10/2026 (paso 4 del plan; `prueba-lib.sh` los comprueba):

1. Al leerse ya no hace nada: no crea directorios en `/tmp` (se reserva el nombre, y cada objetivo crea y borra el suyo), no escribe `SVVERSION` (solo `dist`, y solo con una revisión de verdad) y no baja `distro` de internet (sin el programa, la distribución sale de `/etc/os-release`; la condición de antes miraba otra variable y daba "/bin/distro: No existe").
2. No imprime nada salvo con `SUMMA_CHARLA=1`.
3. `install`: el listado anota los nombres con que quedan los archivos (`changelog.gz`, `Upstream-COPYING`, `Upstream-install.txt`), y los renombres se hacen solo si el archivo

está. `etc/` se instala con sus subdirectorios (`etc/ld.so.conf.d/` de summapack no se instalaba).
4. `dist`: fuera de svn el nombre es `NOMBRE-VERSION.tar.gz` (antes `...RUnversioneddirectory...`); borra su temporal.
5. `tgzbuild`: todo bajo `/usr` (antes la documentación y el listado en `/usr/local`), sin `SVVERSION` entre la documentación, el md5 en `...-AI.tgz.md5`, y borra su temporal.
6. Un proyecto adentro de otro usa su propio `.summa`.
7. **El token de Gemfury ya no está en el archivo**: `uploadgem` lo lee de `../FURY`, como `uploaddeb`. Pero **sigue en la historia del repositorio**: hay que revocarlo en Gemfury.

Quedan:

1. Rutas de una sola persona: `/home/dsa/rpmbuild` en `rpmbuild`, `~/svn/siru` y `~/osc/tabular` en `rpmbuildosc`.
2. `SERVERNAME` fijo en simusol.org aunque el yaml diga otra cosa.
3. En summapack mismo (solo ahí), al leerse crea `/var/cache/summapack` o `~/var/cache/summapack` para las descargas.
4. Varios Makefiles viejos tienen una regla que copia la biblioteca a `../tabular/share/...` (Base, dia-simusol, Plot, simusol-PD, simusol-PD_endesarrollo) o a su propio `share/` (Radia, simusol-SinPlot): no hay que correrla, volvería a haber copias.

O.8. 8. ¿Y para empaquetar calcula y siru? (paso 6, 07/10/2026)

No conviene. calcula y siru, y en total ocho paquetes (mem_util, exprtk, tabular2, unidades, slex y siru en siru; calcula y puyuspa en calcula), se arman con **`hacer-pkg` y `hacer-pkg.inc`**, la otra herramienta de empaquetado del proyecto, más nueva (septiembre de 2026). Por qué no pasarlos a `makefile.lib`:

1. **`debian/` está escrito a mano y es bueno.** El `debuild` de esta biblioteca lo genera con `templatesumma` a partir de una plantilla, y perdería lo que hoy tiene: `Build-Depends` exactos, overrides de lintian, pruebas, el changelog versión por versión (calcula 6.0-38).
2. **Otra forma de versión.** Acá es `BASEVERSION.R<revisión>`; los paquetes de Debian llevan su revisión propia (6.0-38).
3. **`copytree` no sirve para lo que tienen que instalar.** calcula es autotools: se compila, y lo que se instala sale de `make install` (las bibliotecas de los módulos, el programa). siru es Perl, y sus módulos van a `/usr/share/perl5`, no a `PREFIX/lib`, que es adonde `copytree` copia `lib/`.
4. **`hacer-pkg.inc` ya resuelve lo común**, y bien: comprueba que estén las órdenes y los paquetes y dice cómo conseguirlos, arma desde una copia limpia (`svn export`), muestra por qué falló cada paso y copia los `.deb`.

Para qué sirve cada una, entonces:

	`makefile.lib`	`hacer-pkg` + `hacer-pkg.inc`
para	proyectos de scripts y datos (Perl, shell, archivos), que se instalan copiando el árbol	paquetes Debian de verdad, compilados o no, con `debian/` escrito a mano
da	`install` con listado, `dist`, `tgzbuild`, rpm, subir a simusol.org, Gemfury y OBS, `make ayuda`	el `.deb`, armado limpio, con los errores explicados

	`makefile.lib`	`hacer-pkg` + `hacer-pkg.inc`
lo usan	los Simusol, summapack, sit, exprtk (para `dist` e instalar)	calcula, puyuspa, siru, slex, tabular2, exprtk, mem_util, unidades

Un proyecto nuevo que vaya a ser un paquete Debian: `hacer-pkg` con `hacer-pkg.inc`. Uno de scripts que se instala copiando: `makefile.lib`.

`hacer-pkg.inc` tiene una sola implementación en SummaPack. Calcula busca esa copia mediante `SUMMAPACK_DATA`, el árbol vecino de Siru o los datos instalados; no mantiene otra implementación. `hacer-deb.inc` es un enlace compatible.

La publicación actual se hace con `subir-paquetes`; `hacer-pkg` deja `subir.lista` para alimentarlo. Se construyen fuentes tar.gz, DEB, TGZ y RPM cuando está disponible rpmbuild. La tabla anterior conserva la comparación inicial del relevamiento, no una limitación actual a DEB.

O.9. 9. Resumen del plan del 07/10/2026

Hechos los seis pasos: el catálogo, el ejemplo y su prueba, `make ayuda`, que no haga nada al leerse ni hable sin que se le pida y los arreglos de `install`, `dist` y `tgzbuild`, una sola copia instalada, y la decisión de no empaquetar calcula y siru con ella. Quedan los problemas de la sección 7 y revocar el token de Gemfury.

P. Contrato de `sp`

La guía completa de uso está en `README.md`, el manual Unix en `man/sp.1` y el manual Texinfo en `info/sp.texi`; con SummaPack instalado se consultan mediante `man sp` e `info --file=/usr/share/info/sp.info`.

`sp` centraliza en una interfaz común las operaciones de paquetes que el proyecto necesita en Debian, Ubuntu, Fedora, openSUSE y Arch. Detecta la distribución con `/etc/os-release`, delega en el packlet correspondiente y conserva la salida y el código de retorno del gestor. También acepta `--distro` para seleccionar una distribución y `--ensayo` para mostrar los comandos sin ejecutarlos.

P.1. Rosetta, Babel y packlets

La descripción de Rosetta, Babel y Packlets (`https://simusol.org/?Summapack/Rosetta-y-Babel`) separa la traducción de nombres de paquetes de la ejecución de operaciones:

Capa	Responsabilidad en `sp`
Rosetta	Traduce nombres de referencia del diccionario de SummaPack a los nombres que usa cada distribución. No resuelve todavía funciones o proyectos.
Babel	Interpreta acciones como `install`, `remove`, `clean`, `fix`, `search` e `info`, junto con objetos como `DEB`, `RPM`, `TGZ`, `SUMMA`, `SOURCE` y `REPO`.
Packlet	Adapta cada acción al gestor nativo. Sus scripts están en `summapack/sp/packlets/`: APT, DNF, Zypper, Pacman, TGZ y SummaPack.

Los comandos de los packlets incluyen `install`, `remove`, `clean`, `fix`, `search`, `info`, `addrepo`, `delrepo`, `update`, `depclean`, `distupdate` y `refresh`. Cada gestor tiene sus propias limitaciones; por ejemplo, Arch no permite agregar o quitar repositorios con este packlet porque requiere editar `pacman.conf`.

P.2. Interfaz de packlets

`sp` invoca el packlet de la familia como `packlet ACCION [ARGUMENTOS...]`. El primer argumento es una acción normalizada; los argumentos restantes se conservan como elementos separados, sin expansión de shell. Cada packlet usa `common.sh` para imprimir la orden seleccionada y, en modo ensayo, no ejecutarla. `SP_ENSAYO=1` activa ese modo dentro del packlet y `SP_SUMMAPACK_ROOT` le permite localizar los programas y datos de SummaPack desde el árbol de fuentes o una instalación. Una acción desconocida retorna 2; el fallo del comando externo se transmite al llamador.

P.3. Operaciones

Comando `sp`	APT (Debian/Ubuntu)	DNF (Fedora)	Zypper (openSUSE)	Pacman (Arch)
`install PAQUETE...`	`apt-get install`	`dnf install`	`zypper install`	`pacman -Syu --needed`
`install WORLD`	`apt-get upgrade`	`dnf upgrade`	`zypper update`	`pacman -Syu`
`install SYSTEM`	`apt-get dist-upgrade`	`dnf distro-sync`	`zypper dup`	`pacman -Syu`
`install CACHE`	`apt-get update`	`dnf makecache --refresh`	`zypper refresh`	Sincronización completa `pacman -Syu`
`remove PAQUETE...`	`apt-get remove`	`dnf remove`	`zypper remove`	`pacman -R`
`clean WORLD`	`apt-get autoremove`	`dnf autoremove`	No implementado	`pacman -Rns` para dependencias huérfanas
`clean CACHE`	`apt-get clean`	`dnf clean all`	`zypper clean --all`	`pacman -Sc --noconfirm`
`fix WORLD`	`apt-get install -f`	`dnf distro-sync`	`zypper verify`	`pacman -Dk`
`search [all] TEXTO`	`apt-cache search`	`dnf search`	`zypper search`	`pacman -Ss`
`info PAQUETE...`	`apt-cache show`	`dnf info`	`zypper info`	`pacman -Si`
`info CACHE`	`apt-cache stats`	`dnf list installed`	`zypper packages --installed-only`	`pacman -Q`

`install DEB archivo.deb` solo se admite en Debian y Ubuntu; `install RPM archivo.rpm`, en Fedora y openSUSE. `install TGZ archivo.tgz` usa `instalar-tgz` en las cinco familias, y `remove TGZ nombre` retira sus archivos registrados. `install SUMMA proyecto` e `install SOURCE proyecto` delegan en `bajar-paquetes` de SummaPack. `install REPO` y `remove REPO` delegan en el packlet de la distribución.

`buildtools` es una operación de apoyo para preparar entornos de desarrollo. `sp` mantiene una sola lista de dependencias con nombres de referencia Debian, la traduce mediante `diccionariodepaquetes` y entrega los nombres resultantes al packlet del gestor. Los packlets no mantienen listas de paquetes: solo ejecutan la instalación. El refresco de índices sigue la política de la familia.

P.4. Traducción de paquetes

`sp` toma como referencia el nombre Debian del paquete y consulta `diccionariodepaquetes` de SummaPack. Ubuntu tiene una columna propia y, si no hay una entrada específica, puede recurrir a Debian; las demás familias consultan Fedora, SUSE o Arch. Una equivalencia puede contener varios paquetes separados por espacios, por ejemplo `libopenblas-dev:ARCH:openblas cblas`; cada uno se pasa como argumento independiente al gestor. Una equivalencia `-` significa que la dependencia no hace falta en esa distribución: se omite al instalar. El diccionario se busca en el árbol de fuentes y en `/usr/share/summapack/data/` o `/usr/local/share/summapack/data/`; `SUMMAPACK_DICCIONARIO` permite indicar otra ubicación.

Los nombres de paquete se pasan como argumentos separados, nunca como fragmentos de shell. Las operaciones que cambian el sistema usan `sudo` cuando hace falta. `--ensayo` no invoca gestores ni modifica archivos; permite comprobar la selección de packlet y los argumentos en las cinco distribuciones.

P.5. Límites

`install ALL` y los objetos `EBUILD`, `PKG`, `TXZ`, `TBZ`, `SVN`, `CPAN`, `SH`, `LSH` y `COMMAND` no tienen aún implementación. Los repositorios no se administran desde Pacman. `fix` acepta `WORLD` o `SYSTEM`, pero algunas familias usan una operación distinta para aproximar la reparación. En Arch, instalar paquetes desde repositorios siempre sincroniza y actualiza el sistema para evitar una actualización parcial.

Q. sp: interfaz de paquetes de SummaPack

`sp` unifica las operaciones de paquetes que usa el proyecto Simusol. Consulta `bin/distro.sh` para identificar la distribución, traduce nombres de referencia con `diccionariodepaquetes` y delega cada operación en un packlet de APT, DNF, Zypper, Pacman, Portage, URPMI, TGZ o SummaPack. La interfaz sigue el modelo de acciones y objetos descrito en Rosetta, Babel y Packlets (`https://simusol.org/?Summapack/Rosetta-y-Babel`).

Q.1. Instalación y ayuda

`sp` se instala como parte de SummaPack, en `bin/sp`; sus packlets quedan en `share/summapack/sp/packlets`. La guía y el contrato quedan en `share/doc/summapack/sp/`; los manuales se consultan con `man sp` y `info --file=/usr/share/info/sp.info`.

Para una máquina que aún no tiene SummaPack se descarga `https://www.simusol.org/downloads/sp` y se ejecuta `sh sp --para deb summapack` (o `--para fuentes`). Ese archivo contiene el motor de descarga necesario para instalar SummaPack. Una vez instalado, `sp install calcula` es la interfaz habitual. Un usuario sin root instala desde fuentes en su home con `sp --para fuentes --prefijo ~/.local install PROYECTO` (make install con ese prefijo; sp no pide sudo si el directorio es escribible). `packinstall calcula` conserva la sintaxis de scripts antiguos y llama a `sp`; no es otro instalador. Para un proyecto Simusol, `sp` usa internamente `bajar-paquetes`; para paquetes externos usa el gestor de la distribución. `bajar-paquetes` sigue en las fuentes como motor reutilizable, pero ya no es otro archivo de arranque en la raíz de Descargas.

```
sp --help
man sp
info --file=/usr/share/info/sp.info
```

Q.2. Sintaxis

```
sp [--distro DISTRO] [--ensayo] [--sin-upgrade] ACCION [OBJETO] [ARGUMENTOS...]
sp buildtools
sp podman suse-viejo
sp --help
```

Las opciones globales se escriben antes de la acción. `--distro` permite elegir una distribución reconocida o el nombre de una familia. Si se omite, `sp` usa el resultado de `distro.sh`: no mantiene un detector paralelo. `--ensayo` muestra las órdenes seleccionadas y no ejecuta el gestor ni cambia el sistema. `--sin-upgrade` evita la actualización automática de paquetes, pero no la de índices. Los alias `-h`, `--help` y `--ayuda` muestran la ayuda.

Las consultas trabajan con los índices disponibles. Para paquetes de repositorio y herramientas se refrescan los índices vencidos (12 horas). La actualización general se pide con `update` o `distupdate`. Pacman instala con los índices locales y sincroniza y actualiza juntos al pedir `refresh`.

Q.2.1. Paquetes ya instalados y compatibilidad con `packinstall`

`sp` no consulta por separado si cada paquete ya está instalado: traduce su nombre y entrega la solicitud al gestor de la distribución. Es el gestor quien determina si debe instalarlo, actualizarlo o dejar la versión existente. Por eso puede invocar al gestor aunque el paquete ya esté al día; eso no equivale a reinstalarlo.

Implementación	Comprobación previa por paquete	Consecuencia
`viejo/bin/packinstall.pl`	Consultaba `dpkg-query -W` en Debian/Ubuntu, `rpm -q --whatprovides` en Fedora/SUSE/Mandriva, `pacman -Q` en Arch e intentaba cargar los módulos Perl. No instrumentaba esta comprobación en Gentoo/UTUTO.	Si aparecía instalado, omitía llamar al gestor para ese paquete; no comprobaba si había una versión más reciente.
`bin/packinstall.sh` y `sp` actuales	El adaptador `packinstall` reenvía los pedidos a `sp`; `sp` no repite la consulta de instalación para paquetes de la distribución. Sí comprueba si un módulo Perl ya puede cargarse antes de instalarlo.	El gestor decide qué hacer con cada paquete y su versión. `sp` prepara antes los índices y paquetes según el plazo indicado arriba.

Q.3. Acciones

Acción	Uso	Efecto
`install`	`install PAQUETE...`	Instala paquetes traduciendo sus nombres. También acepta los objetos indicados abajo.
`remove`	`remove PAQUETE...`	Desinstala paquetes por nombre. `remove TGZ NOMBRE` retira un paquete registrado por `instalar-tgz`.
`clean`	`clean WORLD` o `clean CACHE`	Elimina dependencias huérfanas o limpia la caché del gestor.
`fix`	`fix`, `fix WORLD` o `fix SYSTEM`	Ejecuta la reparación o sincronización disponible en el gestor de la familia.
`search`	`search [NAMES] TEXTO`	Busca paquetes. `NAMES` es opcional; `search all TEXTO` solicita la búsqueda amplia del gestor cuando existe.
`info`	`info PAQUETE...`	Muestra información de uno o más paquetes. `info WORLD`, `info SYSTEM`, `info ALL` e `info CACHE` consultan estadísticas/listados del gestor.
`buildtools`	`buildtools`	Instala una lista común de herramientas y bibliotecas de desarrollo, traducida para la distribución mediante `diccionariodepaquetes`.
`podman`	`podman ENTORNO`	Descarga una imagen base limpia y la etiqueta como `localhost/simusol-ENTORNO:base`; `ENTORNO` es `ubuntu`, `debian`, `fedora`, `suse` o `arch` seguido de `-viejo` o `-actual`.
`update`, `distupdate`, `refresh`, `depclean`	Sin argumentos	Órdenes de packlet equivalentes a actualizar paquetes, sincronizar distribución, refrescar índices y eliminar dependencias huérfanas.
`addrepo`, `delrepo`	`addrepo REPO`, `delrepo REPO`	Administra repositorios en APT, DNF y Zypper. Pacman no implementa estas acciones.

`WORLD`, `SYSTEM` y `CACHE` también se pueden usar como objetos de `install`: `install WORLD` actualiza paquetes, `install SYSTEM` sincroniza o actualiza la distribución e `install CACHE` refresca índices. `install ALL` está reservado, pero aún no está implementado.

Q.4. Objetos de instalación

Objeto	Forma	Disponibilidad y comportamiento
`PAQ` o `PACKAGES`	`install PAQ NOMBRE...`	Nombres lógicos del diccionario; también se pueden pasar directamente. Los proyectos Simusol conocidos se envían a `bajar-paquetes`.
`DEB`	`install DEB archivo.deb`	Archivo local en distribuciones APT. El archivo debe existir y terminar en `.deb`.
`RPM`	`install RPM archivo.rpm`	Archivo local en Fedora/DNF u openSUSE/Zypper. Debe existir y terminar en `.rpm`.
`TGZ`	`install TGZ archivo.tgz`	Usa `instalar-tgz` en todas las familias; la baja se hace mediante `remove TGZ NOMBRE`.
`SUMMA`	`install SUMMA proyecto`	Descarga e instala un paquete SummaPack mediante `bajar-paquetes`.
`SOURCE`	`install SOURCE proyecto`	Descarga e instala fuentes mediante `bajar-paquetes`.
`PERL`	`install PERL Modulo::Nombre`	Prefiere el paquete de la distribución; si no hay equivalencia, usa cpanm.
`CPAN`	`install CPAN Modulo::Nombre`	Instala directamente con cpanm cuando se quiere omitir el paquete de la distribución.
`REPO`	`install REPO REPO`, `remove REPO REPO`	Añade o elimina un repositorio en APT, DNF o Zypper.

Los objetos `EBUILD`, `PKG`, `TXZ`, `TBZ`, `SVN`, `SH`, `LSH` y `COMMAND` están reconocidos como parte del vocabulario, pero no cuentan todavía con un packlet funcional. `remove` acepta nombres de paquetes instalados, no archivos locales DEB/RPM ni objetos SUMMA/SOURCE.

Q.5. Distribuciones y traducción

Familia	Distribuciones reconocidas (ejemplos)	Packlet
APT	Debian, Ubuntu, Linux Mint, Pop!_OS, Raspbian, Kali, elementary OS, Zorin	`apt.sh`

Familia	Distribuciones reconocidas (ejemplos)	Packlet
DNF	Fedora, RHEL, Red Hat, CentOS, Rocky Linux, AlmaLinux, Oracle Linux, Nobara, OpenMandriva actual	`dnf.sh`
Zypper	SUSE, openSUSE Leap/Tumbleweed, SLES, SLED	`zypper.sh`
Pacman	Arch Linux, Manjaro, EndeavourOS	`pacman.sh`
Portage	Gentoo, UTUTO clásico	`portage.sh` (`emerge`)
URPMI	Mandriva histórica, Mandrake	`urpmi.sh`

Para instalar dependencias, los nombres Debian del diccionario sirven como claves de referencia. Ubuntu, Linux Mint, Pop!_OS, elementary y Zorin consultan primero la columna `UBUNTU` y recurren a `DEBIAN` cuando no hay equivalencia específica. Gentoo consulta `GENTOO`; UTUTO consulta `UTUTO` y luego `GENTOO`; Mandriva consulta `MANDRIVA`; OpenMandriva consulta `OPENMANDRIVA`. En esas cuatro distribuciones, la ausencia de equivalencia es un error: no se entrega al gestor un nombre Debian posiblemente inválido. Las otras familias consultan su columna propia. Una equivalencia `-` indica que la dependencia no existe o no hace falta en esa distribución: se omite al instalar. En otras acciones esa marca produce un error para evitar operar sobre un paquete equivocado.

El diccionario se busca en el árbol de fuentes y en las ubicaciones compartidas de SummaPack. La variable `SUMMAPACK_DICCIONARIO` permite seleccionar otro archivo de diccionario.

Q.6. Privilegios, salida y códigos

Las búsquedas y consultas se ejecutan como usuario. En la instalación desde fuentes vía paquete se compila como usuario y se utiliza `sudo` para las dependencias y para instalar los paquetes resultantes.

La orden completa se imprime en la salida de error como `Ejecutando:` o `Ensayo:`. Los argumentos se entregan al gestor como elementos separados, sin evaluar texto mediante un shell. La salida del gestor se conserva y `sp` propaga su código de salida. Los errores de sintaxis y objetos no implementados retornan 2; una distribución no reconocida también retorna 2. Otros errores de ejecución conservan el código del gestor o el error operativo correspondiente.

En Arch, la preparación usa `pacman -Syu` cuando vence el sello y la instalación posterior usa `pacman -S --needed`. `buildtools` comparte una sola lista de dependencias de referencia; el diccionario puede traducir una de ellas a varios paquetes, como `libopenblas-dev` a `openblas cblas` en Arch. Está pensado para preparar la cadena de desarrollo del proyecto, no para ofrecer una selección interactiva de componentes.

Q.7. Ejemplos

```
# Revisar el comando que se usaría en la máquina actual, sin ejecutarlo.
sp --ensayo install libgsl-dev

# Consultar el nombre y comando que corresponde a Fedora.
sp --distro fedora --ensayo install libgsl-dev
```

```
# Buscar y consultar paquetes.
sp search NAMES gsl
sp info libgsl-dev

# Operaciones de mantenimiento.
sp install CACHE
sp install WORLD
sp clean CACHE
sp fix SYSTEM

# Instalar artefactos y proyectos.
sp install DEB ./paquete.deb
sp install RPM ./paquete.rpm
sp install TGZ ./paquete.tgz
sp remove TGZ nombre-del-paquete
sp install SUMMA dibuja
sp install SOURCE siru

# Inspeccionar las herramientas de desarrollo que se instalarían.
sp --ensayo buildtools
```

Q.8. Desarrollo y pruebas

Los packlets están en `packlets/`; cada uno recibe la acción normalizada como primer argumento y sus operandos después. Las pruebas unitarias usan modo ensayo y no modifican el anfitrión. `tests/test_sp_podman.sh` crea contenedores desechables para revisar la detección y el armado de órdenes en Debian, Ubuntu, Fedora, openSUSE Tumbleweed y Arch; no instala paquetes en el sistema anfitrión.

```
bash -n sp packlets/*.sh tests/*.sh
bash tests/test_sp.sh
tests/test_sp_podman.sh
```

Consulta también `CONTRATO.md` para las responsabilidades de interfaz, la página `sp(1)` para `man sp` y `info/sp.texi` para la fuente del manual Texinfo.

R. Arreglar las instrucciones de instalación para que digan cómo usar `sp`

Pedido del autor, 26/10/2026. Origen: al buscar cómo instalar Calcula, las fuentes locales y el sitio dan instrucciones distintas (`instalar`, `installsumma.sh`, `bajar-paquetes`, dos corridas de `sp`). La única guía correcta y completa es `https://www.simusol.org/downloads/0-README-como-instalar`: bajar `sp` y una sola orden, `sh sp --para FORMATO PROYECTO`, que instala SummaPack, el proyecto y sus dependencias; después queda `sp` instalado. Todo lo demás debe remitir a eso.

R.1. Texto canónico que debe aparecer en todos lados

```
curl -fsSLO https://www.simusol.org/downloads/sp
sh sp --para deb calcula              # binarios Debian/Ubuntu
sh sp --para fuentes-paquete calcula  # construye .deb/.rpm en esta máquina
sh sp --para fuentes calcula          # make install directo en /usr/local
sh sp --prueba --para deb calcula     # muestra el recorrido sin instalar
```

Una sola orden; no hace falta instalar SummaPack antes. Luego: `sp install calcula`, `sp remove calcula`, `sp search gsl`.

R.2. Qué cambiar, dónde

Lugar	Qué dice hoy	Cambio
Sitio, página «Psicro-Calcula» (`simusol.org/?Psicro-Calcula`)	`curl ... installsumma.sh` y `sh installsumma.sh calcula`; `sh installsumma.sh --para fuentes calcula`; `--prueba`	Reemplazar por el texto canónico; `installsumma.sh` ya no es el nombre público
Sitio, «Instalación / Paquetería DEB y RPM»	Solo el repositorio apt de fury (`echo "deb [trusted=yes] https://apt.fury.io/dsaravia/ /"`)	Poner `sp` como método principal y dejar fury como alternativa, o marcarlo como histórico
Sitio, portada, secciones «INSTALAR SIMUSOL Y OTROS PROGRAMAS DESDE SUS FUENTES, PAQUETE POR PAQUETE» y «Ubuntus con versiones mayores o iguales a 20»	No leídas en detalle; por el título describen el camino viejo	Revisar y agregar al principio: «Hoy alcanza con `sp`, ver 0-README-como-instalar»
Sitio, página «Descargas» (`?Descargas`)	Solo dice que los archivos están en downloads y cómo usar `wget -c`	Agregar el texto canónico y el enlace a `downloads/0-README-como-instalar`
`siru/summapack/README`, líneas 47 y 51	`sh sp --para deb summapack` y `sh sp --para fuentes summapack`: solo muestra SummaPack	Agregar el ejemplo con proyecto (`sh sp --para deb calcula`) y aclarar que no son dos pasos
`siru/summapack/PAQUETES.md`, líneas 39, 43–44, 511–512	`bajar-paquetes --para deb --instalar calcula`; `installsumma.sh calcula`	`sp install calcula` con SummaPack instalado; `sh sp --para deb calcula` sin él
`siru/summapack/PAQUETES.md`, líneas 99, 129, 157, 337, 339	Menciones históricas de `installsumma.sh`	Dejar como historia, pero decir una vez que el nombre público actual es `sp`
`siru/summapack/doc/summapack.tex` (14 menciones de `installsumma`), líneas 77–78, 82, 344, 528	El bootstrap del artículo es `installsumma.sh`	Reemplazar por `sp` en los ejemplos; en la parte histórica, nota de que `sp` lo reemplazó
`siru/summapack/doc/web/inicio.html` y `pagina.md`	Mencionan `installsumma` (origen de la página «Summapack» del sitio)	Mismo reemplazo; regenerar con `generar.py` y republicar
`calcula/README`, líneas 12–16 y 26	«ejecute ./configure ... make y make install»; «Página `http://www.psicro.org`»	Primero `sp`; `./configure` como camino de desarrollador; psicro.org redirige a simusol.org
`calcula/calcula-6.0/README`, líneas 39–41 y 58	Igual que el anterior	Igual
`calcula/instalar`, cabecera (líneas 8–19)	`sudo svn/calcula/instalar`, y antes `sudo svn/siru/instalar`	Aclarar que es para quien tiene el working copy; para usuarios, `sp`

Lugar	Qué dice hoy	Cambio
`calcula/CLAUDE.md` y `calcula/PENDIENTE-instalador.md`	Describen `instalar6`, `instalar`, `hacer-deb`, sin `sp`	Una línea al principio: el camino público es `sp`; los demás son internos

R.3. Regla para no volver a divergir

Un solo texto fuente, `downloads/0-README-como-instalar`, y todas las páginas y READMEs lo citan o lo enlazan en lugar de copiar comandos. Cuando cambie `sp`, se cambia ese archivo y se vuelve a subir con `sp subir --bootstrap summapack`.

S. Incidencia resuelta: Slex exigía un mem_util más nuevo que el publicado

Estado al 26/10/2026: SummaPack `2026.2.1.r5350-1` está estable en Descargas con fuente y binarios para Ubuntu, Fedora, openSUSE y Arch; sus catorce archivos pasaron la comprobación MD5. El `sp` público se actualizó. Esta es la descripción histórica del fallo y de la intervención manual que se necesitaba antes de que `sp subir` publicara primero las dependencias propias incompatibles. No hace falta ejecutar de nuevo los pasos de republicación de abajo.

Pedido del autor, 26/10/2026. Para correr en la máquina que tiene la clave ssh del servidor.

S.1. El problema

`sh sp --para fuentes-paquete calcula` falla al construir slex:

```
checking si mem_util tiene mem_util_verbose... no
configure: error: mem_util es vieja (pide la global verbose): instalar la de summapack
     actual
```

Causa: el servidor publica slex 2026.1.0.r5209 (`slex.summa`), cuyo `configure.ac` (línea 24) exige la global `mem_util_verbose`, agregada a mem_util en siru r5178 (23/10/2026). Pero el summapack publicado como estable es 2026.2.1.r4445-1 (20/10/2026), anterior a ese cambio. `sp` instala ese summapack viejo y después slex no compila contra él. No es un problema de la máquina de destino.

S.2. Qué correr, en orden

Con la copia de trabajo de siru al día y sin cambios pendientes:

```
cd ~/svn/siru && svn update && svn status summapack     # debe salir vacío
cd ~/svn/siru/summapack
sp paquete summapack          # arma con hacer-pkg: tar.gz, .summa con versión (.r5xxx
    ), paquetes
sp subir summapack            # sube lo armado como UNSTABLE a SUBIR_A del .summa
~/svn/siru/summapack/sp/sp --prueba --para fuentes-paquete --inestable slex   #
    opcional: probar la inestable
sp estable summapack          # lo subido pasa a estable
sp subir --bootstrap summapack   # vuelve a publicar downloads/sp con el bajar-
    paquetes actual
```

Si `sp` no está instalado en esa máquina, usar los scripts del repositorio: `~/svn/siru/summapack/hacer-pkg` y luego `subir-paquetes $(cat ~/svn/siru/subir.lista)` y `subir-paquetes --estable summapack` (PAQUETES.md, sección «Instalar en otra máquina» y «sp subir»).

S.3. Verificación

En cualquier máquina:

```
curl -fsSL https://www.simusol.org/downloads/summapack.summa | grep ^VERSION=
```

debe mostrar una revisión mayor o igual a r5178. Después, en la máquina donde falló:

```
cd ~ && curl -fsSLO https://www.simusol.org/downloads/sp && sh sp --para fuentes-
    paquete calcula
```

S.4. Para que no se repita

Cuando un proyecto empiece a exigir algo nuevo de mem_util (o de otra dependencia propia), publicar primero la dependencia como estable y recién después el proyecto. Alternativa: que `slex.summa` declare la versión mínima de summapack en DEPENDS, así `sp` la actualiza antes de compilar.

Referencias

[1] D. Saravia. *Manifiesto Anti Distros*, versión 0.60, 2005. `http://docs.hipatia.info/distros/`. Copia en `doc/UTUTO-ANTIDISTROS.html`; el agregado de la capacidad Ajolote, en el apéndice E.

[2] D. Saravia. *Rosetta, Babel y Packlets*, parte del proyecto Sumapack. Copia en `doc/UTUTO-Rosetta-Babel-y-Packlets.html`; la página de Sumapack, en `doc/UTUTO-SummaPack.html`.

[3] D. Saravia. *Sumapack: herramientas para la compilación, empaquetado e instalación de software libre en diferentes arquitecturas, sistemas operativos y distribuciones GNU/Linux*. 40 JAIIO, Jornadas de Software Libre, 2011, pp. 41–59. Copia en `doc/738.pdf`.

[4] el apéndice N: todo el recorrido, sólo con órdenes de `sp`.

[5] el apéndice Ñ: la referencia de uso, que se mantiene junto al código.