OPKG Package Manager

The `opkg` utility (an ipkg fork) is a lightweight package manager used to download and install OpenWrt packages from local package repositories or ones located in the Internet.
GNU/Linux users already familiar with `apt-get`, `aptitude`, `pacman`, `yum`, etc. will recognize the similarities. It also has similarities with NSLU2's Optware [http://www.nslu2-linux.org/wiki/Optware/], also made for embedded devices. OPKG is however a full package manager for the root file system, instead of just a way to add software to a seperate directory (e.g. `/opt`). This also includes the possibility to add kernel modules and drivers. OPKG is sometimes called *Entware*, but this is mainly to refer to the Entware repository [http://entware.wl500g.info] for embedded devices.

`Opkg` attempts to resolve dependencies with packages in the repositories - if this fails, it will report an error, and abort the installation of that package.

Missing dependencies with third-party packages are probably available from the source of the package.
To ignore dependency errors, pass the `--force-depends` flag.

Please note: If you are using a snapshot / trunk / bleeding edge version, `opkg install <pkg>` may fail, if the package in the repository is for a newer kernel version than the kernel version on your flash. In this case, you will get the error message *"Cannot satisfy the following dependencies for..."*
See also OpenWrt FAQ

Invocation

```
opkg must have one sub-command argument:
usage: opkg [options...] sub-command [arguments...]
where sub-command is one of:
```

You can use glob patterns [https://en.wikipedia.org/wiki/Glob_(programming)].

Package Manipulation

`update`	Update list of available packages This simply retrieves a file like this one: example [http://downloads.openwrt.org/snapshots/trunk/ar71xx/packages/Packages], for your OpenWrt installation and stores it on your RAM partition [https://en.wikipedia.org/wiki/tmpfs] under `/tmp/opkg-lists/snapshots`. Currently this occupies about 1,3 MiB of space. OPKG needs the content of this file in order to install or upgrade packages or to print info about them. And the content of the file needs to represent the current available packages in the repository. You can safely delete this file anytime to free up some RAM, just don't forget to fetch a new one, before trying to install a package.
`upgrade` `<pkgs>`	Upgrade packages To upgrade a group of packages, run `opkg upgrade packagename1 packagename2`. A list of upgradeable packages can be obtained with the `opkg list-upgradable` command. Upgrading packages is generally **not** recommended for most users, because a typical OpenWrt system stores the base system in a read-only SquashFS partition. And while the upgrade process works just fine, it uses far more space than a default installation as the base packages are duplicated in the base SquashFS partition and the user JFFS2 partition. Thus, instead of upgrading, reflashing OpenWrt with a newer firmware image is recommended. Of course, upgrading packages installed afterwards does not have this drawback. Keep in mind though that for OpenWrt releases upgrading is for the most part not possible, since there is nothing to upgrade without changing the package repository. This is because the package repositories for OpenWrt's releases are generally not updated. However, the package repository in the trunk snapshots are updated by the build bots to new versions very often, as this is where the packages are updated, like the OpenWrt builds themselves. Note however that for kernel packages updating can be a risky business as it may brick the device if the trunk build kernel is incompatible with the new upgraded kernel package. You should therefore only upgrade non-kernel packages.

`install` `<pkgs|FQDN>`	Install package(s)
	Examples:
	`opkg install hiawatha`
	`opkg install http://downloads.openwrt.org/snapshots/trunk/ar71xx/packages/hiawatha_7.7-2_ar71xx.ipk`
	`opkg install /tmp/hiawatha_7.7-2_ar71xx.ipk`

`configure` `<pkgs>`	Configure unpacked package(s)
`remove` `<pkgs|globp>`	Remove package(s)

flag <flag> <pkgs>	Flag one or multiple package(s). Only one flag per invocation is allowed. Available flags: hold • noprune • user • ok • installed • unpacked

Informational Commands

list [pkg|globp]	List available packages `Package name - Version - Description` The Description can contain line breaks, so using merely grep is inapt since grep is line-based.
list-installed	List installed packages
list-upgradable	List installed and upgradable packages
list-changed-conffiles	List user modified configuration files

files <pkg>	List files belonging to <pkg>. The package has to be already installed for this to work. Example:
	`opkg files asterisk18`
	`Package asterisk18 (1.8.4.4-1) is installed on root and has the following files:`
	`/usr/lib/asterisk/modules/res_rtp_multicast.so`
	`/usr/lib/asterisk/modules/codec_ulaw.so`
	`/etc/asterisk/features.conf`
	`/usr/lib/asterisk/modules/format_wav_gsm.so`
	`/usr/lib/asterisk/modules/app_macro.so`
	`/usr/lib/asterisk/modules/chan_sip.so`
	`/usr/lib/asterisk/modules/app_dial.so`
	`/usr/lib/asterisk/modules/app_playback.so`
	`/usr/lib/asterisk/modules/format_gsm.so`
	`/usr/lib/asterisk/modules/func_callerid.so`
	`/usr/lib/asterisk/modules/func_timeout.so`
	`/etc/asterisk/asterisk.conf`
	`/etc/asterisk/modules.conf`
	`/usr/lib/asterisk/modules/format_wav.so`
	`/etc/asterisk/extensions.conf`
	`/etc/init.d/asterisk`
	`/etc/asterisk/manager.conf`
	`/usr/lib/asterisk/modules/res_rtp_asterisk.so`
	`/etc/asterisk/logger.conf`
	`/etc/asterisk/rtp.conf`
	`/usr/lib/asterisk/modules/codec_gsm.so`
	`/etc/asterisk/indications.conf`
	`/usr/lib/asterisk/modules/func_strings.so`
	`/usr/lib/asterisk/modules/app_echo.so`
	`/usr/lib/asterisk/modules/format_pcm.so`
	`/etc/asterisk/sip_notify.conf`
	`/etc/asterisk/sip.conf`
	`/etc/default/asterisk`
	`/usr/sbin/asterisk`
	`/usr/lib/asterisk/modules/pbx_config.so`
	`/usr/lib/asterisk/modules/func_logic.so`

search <file|globp>	List package providing <file>

info [pkg|globp]	Display all info for <pkg>
	`Package: horst`
	`Version: 2.0-rc1-2`
	`Depends: libncurses`
	`Provides:`
	`Status: install user installed`
	`Section: net`
	`Architecture: ar71xx`
	`Maintainer: Bruno Randolf <br1@einfach.org>`
	`MD5Sum: 378cea9894ec971c419876e822666a6a`
	`Size: 19224`
	`Filename: horst_2.0-rc1-2_ar71xx.ipk`
	`Source: feeds/packages/net/horst`
	`Description: [horst] is a scanning and analysis tool for 802.11 wireless networks and`
	`especially IBSS (ad-hoc) mode and mesh networks (OLSR).`
	Note1: The size is the size of the gzip compressed tar archive. At installation package gets un-tared and decompressed, but then again JFFS2 uses compression itself.
	Note2: Since the compression of JFFS2 is transparent, commands like `ls` will always report the size of the uncompressed file.

status [pkg|globp]	Display all status for <pkg>
download <pkg>	Download <pkg> to current directory
compare-versions <v1> <op> <v2>	Compare versions `v1` and `v2` using the operators `<=`, `<`, `>`, `>=`, `=`, `<<` or `>>`

print-architecture	List installable package architectures
whatdepends [-A] [pkgname|pat]+	This only works for installed packages. So if you would like to know, how much storage a package and all of it's dependencies would need, at the moment, you will have to piece this information together with the `info`-option.
whatdependsrec [-A] [pkgname|pat]+	This only works for installed packages. So if you would like to know, how much storage a package and all of it's dependencies would need, at the moment, you will have to piece this information together with the `info`-option.
whatprovides [-A] [pkgname|pat]+	
whatconflicts [-A] [pkgname|pat]+	
whatreplaces [-A] [pkgname|pat]+	

Options

Option	Long	Description
`-A`		Query all packages not just those installed
`-V[<level>]`	`--verbosity[=<level>]`	Set verbosity level to <level>. Available verbosity levels: 0 errors only 1 normal messages (default) 2 informative messages 3 debug 4 debug level 2
`-f <conf_file>`	`--conf <conf_file>`	Use <conf_file> as the opkg configuration file. Default is `/etc/opkg.conf`
	`--cache <directory>`	Use a package cache
`-d <dest_name>`	`--dest <dest_name>`	Use <dest_name> as the the root directory for package installation, removal, upgrading. <dest_name> should be a defined dest name from the configuration file, (but can also be a directory name in a pinch).
`-o <dir>`	`--offline-root <dir>`	Use <dir> as the root directory for offline installation of packages.
	`--add-arch <arch>:<prio>`	Register architecture with given priority
	`--add-dest <name>:<path>`	Register destination with given path
Force Options		
	`--force-depends`	Install/remove despite failed dependencies
	`--force-maintainer`	Overwrite preexisting config files
	`--force-reinstall`	Reinstall package(s)
	`--force-overwrite`	Overwrite files from other package(s)
	`--force-downgrade`	Allow opkg to downgrade packages
	`--force-space`	Disable free space checks
	`--force-checksum`	Ignore checksum mismatches
	`--force-postinstall`	Run postinstall scripts even in offline mode
	`--noaction`	No action – test only
	`--download-only`	No action – download only
	`--nodeps`	Do not follow dependencies
	`--force-removal-of-dependent-packages`	Remove package and all dependencies
	`--autoremove`	Remove packages that were installed automatically to satisfy dependencies
`-t`	`--tmp-dir`	Specify tmp-dir.

Examples

To install a package run the following commands. List of available packages is lost upon reboot, so make sure to update the list before trying to install a package

```
opkg update
opkg install <package>
```

To search

- `opkg list` will display only `Package name – Version – Description`
- `opkg info` will display all available information.

You can make use of glob pattern [https://en.wikipedia.org/wiki/Glob_(programming)]s directly and also write a little shell script to use Regular expression [https://en.wikipedia.org/wiki/Regular expression]s and otherwise further process information. Use a pipe (|) and `grep` or `awk` or `sed` to filter that output:

- `opkg list | grep pattern`
- `opkg list | awk '/pattern/ {print $0}`
- `opkg info kmod-ipt-* | awk '/length/ {print $0}'`
- `opkg list-installed | awk '{print $1}' | sed ':M;N;$!bM;s#\n# #g'`
- `var="packagename1 packagename2 packagename2"; for i in $var; do opkg info $i; done;`
- `opkg depends dropbear` doesn't work either.

Configuration

Adjust Repositories

The only configuration file is `/etc/opkg.conf`. It could look like this:

```
src/gz snapshots http://downloads.openwrt.org/snapshots/trunk/ar71xx/packages
dest root /
dest ram /tmp
lists_dir ext /var/opkg-lists
option overlay_root /overlay
```

Local Repositories

You can configure opkg to fetch the packages locally:

```
src/gz local file:///path/to/packagesDirectory
```

Barrier_breaker uses multiple repositories, where every repository requires a unique identifier. It is logical to use their original names, e.g.:

```
...
src/gz base file:///path/to/packages/directory/packages/base
src/gz luci file:///path/to/packages/directory/packages/luci
src/gz packages file:///path/to/packages/directory/packages/packages
src/gz oldpackages file:///path/to/packages/directory/packages/oldpackages
... etc ...
```

Live example:

```
r=44685
search="http://downloads.openwrt.org/snapshots/trunk/ar71xx/generic"
replace="file:///mnt/sdcard/shared/users/www/r$r"
sed -i -e "s!$search!$replace!" /etc/opkg.conf
```

Share for a second router:

```
ln -s /mnt/sdcard/shared/users/www /www/pendrive
```

In the second router:

```
r=44685
search="downloads.openwrt.org/snapshots/trunk/ar71xx/generic"
replace="192.168.1.1/pendrive/r$r"
sed -i -e "s!$search!$replace!" /etc/opkg.conf
```

Adjust Architectures

By default, *opkg* only allows packages with the architecture `all` (= architecture independant) and the architecture of the installed target. In order to source packages from a foreign, but compatible target, the list of allowed architectures can be overrided in opkg.conf with the use of `arch` options:

```
arch all 100
arch brcm4716 200
arch brcm47xx 300
```

This example would allow installing `brcm47xx` (= family of SoCs) packages on the `brcm4716` (a specific SoC) target. The number specifies a priority index which is used by `opkg` to determine which package to prefer in case it is available in multiple architectures.

Proxy support

To use opkg through a proxy, add the following to /etc/opkg.conf:

```
option http_proxy http://proxy.example.org:8080/
option ftp_proxy ftp://proxy.example.org:2121/
```

Use the options below to authenticate against the proxy server:

```
option proxy_username xxxx
option proxy_password xxxx
```

The authentication may fail due to the limitations of busybox wget. Try passing the username and password as part of the url in this case.

```
option http_proxy http://username:password@proxy.example.org:8080/
option ftp_proxy http://username:password@proxy.example.org:2121/
```

Installation Destinations

Extroot

Use extroot and you are done with it. No further configuration is necessary.

Mount Point

One very useful feature of opkg, which may be unfamiliar to those used to tools such as the apt family, is the ability to specify a destination for any package installation.

(!) Many packages are not relocatable and will fail to install cleanly when installed to a non-root location! LuCI for example will fail to find its modules and will not work without manual fixes! Use Extroot instead!
(!) Don't expect this solution to work out-of-the-box, most packages will need additional symlinks or hacks to correctly work under the changed path!

The default opkg.conf actually contains three destinations:

```
dest root /
dest ram /tmp
dest mnt /mnt
```

The format of destination lines is simply the keyword dest, followed by a name for this destination (this can be anything), followed by a filesystem location. Any destination that has been thus configured can then be specified on the opkg command line like this:

```
opkg install somepackage -d destination_name
```

The *dest* argument must refer to one of the defined destinations in /etc/opkg.conf, e.g. -d ram to install packages to /tmp/.

If you want to install kernel modules on any other destination than root, you might want to read this first: https://dev.openwrt.org/ticket/10739 [https://dev.openwrt.org/ticket/10739]

Detailed Instructions

First mount an external filesystem, see Mounting Filesystems for help with that. Then edit /etc/opkg.conf:

- Add the line dest usb /opt to the bottom of the file
- Execute the following command (assuming that /mnt/sda1 is the path to the mount point of your external filesystem):

  ```
  ln -s /mnt/sda1 /opt
  ```

- If you installed packages to tmp folder via opkg -d ram then you need to add new binary and library paths:

  ```
  export PATH=$PATH:/tmp/bin:/tmp/sbin:/tmp/usr/bin:/tmp/usr/sbin
  export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/tmp/lib:/tmp/usr/lib
  ```

- Edit /etc/profile and add the new mount point to your paths variables:

  ```
  export PATH=<current default path>:/opt/bin:/opt/sbin:/opt/usr/bin:/opt/usr/sbin
  export LD_LIBRARY_PATH=<current default LD library path>:/opt/lib:/opt/usr/lib
  ```

- From here, you should be able to install new packages to your new mount point as follows:

  ```
  opkg update
  opkg -dest usb install asterisk14  # or whatever else you want…
  ```

- If you install packages to external filesystems that have a startup script under */etc/init.d* and (if enabled) under */etc/rc.d* you need to set a sysmlink to */etc/init.d*, e.g.:

  ```
  ln -s /usb/etc/init.d/openvpn /etc/init.d/openvpn
  ```

- Libraries installed along with those packages are also installed to the external filesystem. This causes the programm not to start during bootup. You need to manually set the LD_LIBRARY_PATH in each 'external' startup script:

```
export LD_LIBRARY_PATH=/lib:/usr/lib:/tmp/lib:/tmp/usr/lib:/usb/lib:/usb/usr/lib
```

Alternative Instructions

1. mount external partition, see Mounting Filesystems
2. change `/etc/opkg.conf` and either add a new destination to desired mount point, or
3. modify dest root to point to your usb drive
4. copy contents of `/usr/lib/opkg` to your mount point, e.g. USB drive
5. map directories where it is likely that they'll receive lots of files to your usb drive, for example with mount –bind during startup. `/etc/rc.local` could be one place where to map the directories.

example:

```
mount --bind /mnt/usb/root /root
mount --bind /mnt/usb/usr/local /usr/local
mount --bind /mnt/usb/home  /home
```

Libraries

For packages which contain libraries, and are installed on external drive, those libraries need to be found. Installing `ldconfig`, and listing the additional library paths in `/etc/ld.so.conf` will take care of that. ldconfig must be executed after new libraries have been installed, and it can be installed to external drive. Executing it once during boot may be a good idea. Maybe `/etc/rc.local`.

Kernel Modules

Kernel modules that are installed to a non-standard location may not be properly automatically inserted into the kernel when required, and manual insertion may be necessary. For example, after installing libdevmapper under /mnt, insert the modules thus:

```
insmod /mnt/lib/modules/2.6.36.4/dm-mod.ko
```

You could install daemons and "core" services on the internal flash and install optional packages to external mount point. During boot, the external directories with executables (<mountpoint>/[bin,sbin,usr/bin,usr/sbin] are added to PATH if mount of external drive took place. That way the router continues to provide all relevant services, even with external storage disconnected.

Problems arose with these packages, when installing to external drive:

- `file`
  It installs its dependency libmagic to external drive too, but looks for that lib on root device.
  Fix: a script with be name "file" in a directory in PATH which is searched before the directory which contains the executable "file", which calls file (full path) with option -m and path to the libmagic:

  ```
  #!/bin/sh
  /mnt/usb/usr/bin/file -m /mnt/usb/usr/share/file/magic "$@"
  ```

  after saving the script, and making it executable (chmod +x /path/of/script/file), executing once "hash -r" may or may not be needed.
- `netcat`
  Not really a problem, but the link nc (busybox?) was still pointing to a file with wrong location.
  Moving the nc link to external drive (to the dir containing netcat) fixes that.
- `nfsd`
  `/etc/init.d/nfsd` contains hardcoded paths to executables on `/usr/sbin`
  Fix: edit `/etc/init.d/nfsd`, by changing those paths to updated location.

- `lvm2`
  `/etc/init.d/lvm2` contains hardcoded paths to executables on `/sbin`
  Fix: edit `/etc/init.d/lvm2`, by changing these paths to updated location.

More information on installing and using `ldconfig` can be found in this article [https://forum.openwrt.org/viewtopic.php?pid=117496#p117496].

Example Creating Links to Files from the Root File System to Make Programs Work

Some programs need additional config files to run, and you will have to create soft links between the root filesystem and the external storage one (e.g. USB). As an example, to successfully run Midnight Commander after installing it on a USB stick, you must run:

```
ln -s $USB/usr/share/terminfo/ /usb/share/
ln -s $USB/etc/mc /etc/mc
```

Troubleshooting

Out of space

If *opkg* runs out of space, it usually fails to recover cleanly leaving dangling lock files in place resulting in a *`Could not obtain administrative lock`* error. The lock file can be deleted by issuing the `rm /usr/lib/opkg/lock` command.

Additionally, *opkg* may not remove the files it was installing.
One way to do this is get a list of the files it was installing, then delete them.

Replace the url with the appropriate package.

```
(cd /; \
 wget -qO- http://downloads.openwrt.org/snapshots/trunk/ar7/packages/6in4_10-1_all.ipk | \
 tar -Oxz ./data.tar.gz | tar -tz | xargs rm)
```

However, the above line does not delete the dependencies that were installed along with the package responsible. Also, it leaves empty directories around. The script below intends to fix those.

```
#!/bin/sh
#takes one argument/parameter: the name of the package which didn't install correctly and should be removed along with its dependencies
#example: ./opkgremovepartlyinstalledpackage.sh pulseaudio-daemon

#get list of all packages that would be installed along with package x
PACKAGES=`opkg --force-space --noaction install $1 | grep "http:" | cut -f 2 -d ' ' | sed 's/\.$//'`
opkg update
for i in $PACKAGES
do
        LIST=`wget -qO- $i | tar -Oxz ./data.tar.gz | tar -tz | sort -r | sed 's/^./\/overlay\/upper/'`
        for f in $LIST
        do
                if [ -f $f ]
                then
                        echo "Removing file $f"
                        rm -f $f
                fi
                if [ -d $f ]
                then
                        echo "Try to remove directory $f (will only work on empty directories)"
                        rmdir $f
                fi
        done
done
echo "You may need to reboot for the free space to become visible"
```

Save it as `opkgremovepartlyinstalledpackage.sh` somewhere in your OpenWrt box, set it as an executable with `chmod +x ./opkgremovepartlyinstalledpackage.sh` and you can execute it by doing `./opkgclean.sh <package-name>` .

Extras

For colorize opkg output, you can use http://pastie.org/5464938 [http://pastie.org/5464938].

Commands

To find installed pkgs of a specific install target (ex. USB) (**DRAWBACK!!! if any update availlable, it will update the package, just be warned!!!**):

```
for pkg in `opkg list-installed | sed -e "s/^([0-9A-Za-z\-]+) - .*$/\1/p" -n`; do opkg install $pkg; done | grep -i installed\ in\ <TARGET>
```

Notes

Since Trunk r23173 [https://dev.openwrt.org/changeset/23173] respectively Backfire r23206 [https://dev.openwrt.org/changeset/23206] the kernel and kmod packages are flagged as *hold*.
The `opkg upgrade` command won't attempt to update them anymore.

- packages
- Pacman Rosetta [https://wiki.archlinux.org/index.php/Pacman_Rosetta]