pkgbase: Are we there yet?

Emmanuel Vadot
manu@FreeBSD.org

FreeBSD

EuroBSDCon
Lillehammer, Norway
September 19 – 22, 2019
Who am I

- Emmanuel Vadot (manu@FreeBSD.Org)
- FreeBSD user since 2004
- FreeBSD src commiter since 2016
- FreeBSD ports commiter since 2018
- Freelance developer
What is pkgbase?

- Using pkg(8) for packaging and updating base
What is pkgbase?

- Using pkg(8) for packaging and updating base
- pkg(8) is the default package manager since FreeBSD 10.0
What is pkgbase?

- Using `pkg(8)` for packaging and updating base
- `pkg(8)` is the default package manager since FreeBSD 10.0
- Splits base into multiple packages
What is pkgbase?

- Using pkg(8) for packaging and updating base
- pkg(8) is the default package manager since FreeBSD 10.0
- Splits base into multiple packages
- Started in 2015 (yeah ...) by bapt@
Goals

- Binary upgrades for RELEASE, STABLE and CURRENT
Goals

- Binary upgrades for RELEASE, STABLE and CURRENT
- Fine grain installation (no sendmail, no toolchain etc ... )
Goals

▶ Binary upgrades for RELEASE, STABLE and CURRENT
▶ Fine grain installation (no sendmail, no toolchain etc ...)
▶ Let pkg(8) deal with conf file updates
Goals

- Binary upgrades for RELEASE, STABLE and CURRENT
- Fine grain installation (no sendmail, no toolchain etc ...)
- Let pkg(8) deal with conf file updates
- Allow developers to provide package for users to test
Goals (2)

- In the build system (make packages)
Goals (2)

- In the build system (make packages)
- Run as user
Goals (2)

- In the build system (make packages)
- Run as user
- Cross arch creation of packages
Goals (2)

- In the build system (make packages)
- Run as user
- Cross arch creation of packages
- I want people to create FreeBSD “distros”
How packages are generated

- Install a “fake” root during target worldstage/kernelstage

- Uses -DNO ROOT and METALOG (mtree file)

- Add tags into the METALOG with the destination package

- Defaults to FreeBSD-utilities package

- Makefiles can override the package with PACKAGE=XXX

- ucls (package definition) are in release/packages

- plist (package content) automatically generated

- Package and repository are created by make packages target
How packages are generated

- Install a “fake” root during target worldstage/kernelstage
- Uses -DNO_ROOT and METALOG (mtree file)
How packages are generated

- Install a “fake” root during target worldstage/kernelstage
- Uses -DNO_ROOT and METALOG (mtree file)
- Add tags into the METALOG with the destination package
How packages are generated

- Install a “fake” root during target worldstage/kernelstage
- Uses -DNO_ROOT and METALOG (mtree file)
- Add tags into the METALOG with the destination package
- Defaults to FreeBSD-utilities package
How packages are generated

- Install a “fake” root during target worldstage/kernelstage
- Uses -DNO_ROOT and METALOG (mtree file)
- Add tags into the METALOG with the destination package
- Defaults to FreeBSD-utilities package
- Makefiles can override the package with PACKAGE=XXX
How packages are generated

- Install a “fake” root during target worldstage/kernelstage
- Uses -DNO_ROOT and METALOG (mtree file)
- Add tags into the METALOG with the destination package
- Defaults to FreeBSD-utilities package
- Makefiles can override the package with PACKAGE=XXX
- ucls (package definition) are in release/packages
How packages are generated

- Install a “fake” root during target worldstage/kernelstage
- Uses -DNO_ROOT and METALOG (mtree file)
- Add tags into the METALOG with the destination package
- Defaults to FreeBSD-utilities package
- Makefiles can override the package with PACKAGE=XXX
- ucls (package definition) are in release/packages
- plist (package content) automatically generated
How packages are generated

- Install a “fake” root during target worldstage/kernelstage
- Uses -DNO_ROOT and METALOG (mtree file)
- Add tags into the METALOG with the destination package
- Defaults to FreeBSD-utilities package
- Makefiles can override the package with PACKAGE=XXX
- ucls (package definition) are in release/packages
- plist (package content) automatically generated
- Package and repository are created by make packages target
How base is split

- Current split isn’t final
- FreeBSD-kernel-$kernconf: Each Kernel in its own package (based on the config)
- FreeBSD-bootloader contain bootloaders and configuration files (lua or forth)
- FreeBSD-clibs contain the C runtime (ld-elf.so.1, libc, libthr etc ...)
- FreeBSD-runtime contain everything for booting to single user and repair an installation
- FreeBSD-rc contain the rc subsystem
- FreeBSD-utilities is the default package so contain a lot of different thing

- Some stuff will be moved out of it
How base is split

▶ Current split isn’t final
How base is split

- Current split isn’t final
- FreeBSD-kernel-$kernconf : Each Kernel in its own package (based on the config)
How base is split

- Current split isn’t final
- FreeBSD-kernel-$kernconf : Each Kernel in its own package (based on the config)
- FreeBSD-bootloader contain bootloaders and configuration files (lua or forth)
How base is split

- Current split isn’t final
- FreeBSD-kernel-$kernconf : Each Kernel in its own package (based on the config)
- FreeBSD-bootloader contain bootloaders and configuration files (lua or forth)
- FreeBSD-clibs contain the C runtime (ld-elf.so.1, libc, libthr etc ...)
- FreeBSD-utilities is the default package so contain a lot of different thing
- Some stuff will be moved out of it
How base is split

» Current split isn’t final
» FreeBSD-kernel-$kernconf : Each Kernel in its own package (based on the config)
» FreeBSD-bootloader contain bootloaders and configuration files (lua or forth)
» FreeBSD-clibs contain the C runtime (ld-elf.so.1, libc, libthr etc ...)
» FreeBSD-runtime contain everything for booting to single user and repair an installation
How base is split

- Current split isn’t final
- FreeBSD-kernel-$kernconf: Each Kernel in its own package (based on the config)
- FreeBSD-bootloader contain bootloaders and configuration files (lua or forth)
- FreeBSD-clibs contain the C runtime (ld-elf.so.1, libc, libthr etc ...)
- FreeBSD-runtime contain everything for booting to single user and repair an installation
- FreeBSD-rc contain the rc subsystem
How base is split

- Current split isn’t final
- FreeBSD-kernel-$kernconf: Each Kernel in its own package (based on the config)
- FreeBSD-bootloader contain bootloaders and configuration files (lua or forth)
- FreeBSD-clibs contain the C runtime (ld-elf.so.1, libc, libthr etc ...)
- FreeBSD-runtime contain everything for booting to single user and repair an installation
- FreeBSD-rc contain the rc subsystem
- FreeBSD-utilities is the default package so contain a lot of different thing
How base is split

- Current split isn’t final
- FreeBSD-kernel-$kernconf : Each Kernel in its own package (based on the config)
- FreeBSD-bootloader contain bootloaders and configuration files (lua or forth)
- FreeBSD-clibs contain the C runtime (ld-elf.so.1, libc, libthr etc ...)
- FreeBSD-runtime contain everything for booting to single user and repair an installation
- FreeBSD-rc contain the rc subsystem
- FreeBSD-utilities is the default package so contain a lot of different thing
- Some stuff will be moved out of it
How base is split (cont.)

▶ Every package is split with -debug -development -profile

▶ On 64 bits arch with 32 bits support some -lib32 packages are created

▶ Every lib/programs from contrib/ in their own package (Easier for SA/EN)

▶ FreeBSD-tests contain all the testsuite (should we put kyua there ?)

▶ Other packages are application or lib specifics, e.g.:

  FreeBSD-bluetooth/FreeBSD-wpa/FreeBSD-ssh/FreeBSD-libarchive...

▶ Will continue to move things out of utilities when it make sense (nfs ? kerberos ?)
How base is split (cont.)

- Every package is split with `-debug -development -profile` package

- On 64 bits arch with 32 bits support some `-lib32` packages are created

- Every lib/programs from contrib/ in their own package (Easier for SA/EN)

- FreeBSD-tests contain all the testsuite (should we put kyua there?)

- Other packages are application or lib specifics, e.g.: FreeBSD-bluetooth/FreeBSD-wpa/FreeBSD-ssh/FreeBSD-libarchive...

- Will continue to move things out of utilities when it make sense (nfs ? kerberos ?)
How base is split (cont.)

- Every package is split with -debug -development -profile package
- On 64 bits arch with 32 bits support some -lib32 packages are created

FreeBSD-tests contain all the testsuite (should we put kyua there ?)

Other packages are application or lib specifics, e.g.:
FreeBSD-bluetooth/FreeBSD-wpa/FreeBSD-ssh/FreeBSD-libarchive...

Will continue to move things out of utilities when it make sense (nfs ? kerberos ?)
How base is split (cont.)

- Every package is split with -debug -development -profile package
- On 64 bits arch with 32 bits support some -lib32 packages are created
- Every lib/programs from contrib/ in their own package (Easier for SA/EN)
How base is split (cont.)

- Every package is split with -debug -development -profile package
- On 64 bits arch with 32 bits support some -lib32 packages are created
- Every lib/programs from contrib/ in their own package (Easier for SA/EN)
- FreeBSD-tests contain all the testsuite (should we put kyua there ?)
How base is split (cont.)

- Every package is split with -debug -development -profile package
- On 64 bits arch with 32 bits support some -lib32 packages are created
- Every lib/programs from contrib/ in their own package (Easier for SA/EN)
- FreeBSD-tests contain all the testsuite (should we put kyua there ?)
- Other packages are application or lib specifics, e.g. : FreeBSD-bluetooth/FreeBSD-wpa/FreeBSD-ssh/FreeBSD-libarchive
  ...

FreeBSD
How base is split (cont.)

- Every package is split with `-debug -development -profile` package
- On 64 bits arch with 32 bits support some `-lib32` packages are created
- Every lib/programs from contrib/ in their own package (Easier for SA/EN)
- FreeBSD-tests contain all the testsuite (should we put kyua there ?)
- Other packages are application or lib specifics, e.g. : FreeBSD-bluetooth/FreeBSD-wpa/FreeBSD-ssh/FreeBSD-libarchive
  ...
- Will continue to move things out of utilities when it make sense (nfs ? kerberos ?)
Number of packages

- It apparently matters to some people
Number of packages

- It apparently matters to some people
- It matters to me only for time spent installing/upgrading
Number of packages

- It apparently matters to some people
- It matters to me only for time spent installing/upgrading
- Total: 392 (529MB with xz compression)
Number of packages

- It apparently matters to some people
- It matters to me only for time spent installing/upgrading
- Total: 392 (529MB with xz compression)
- Current count without -debug/-development/-profile: 118 (158MB with xz compression)

Number of packages will only go up starting now
Number of packages

- It apparently matters to some people
- It matters to me only for time spent installing/upgrading
- Total: 392 (529MB with xz compression)
- Current count without -debug/-development/-profile: 118 (158MB with xz compression)
- Current count without -debug/-development/-profile/-lib32: 80 (150MB with xz compression)
Number of packages

- It apparently matters to some people
- It matters to me only for time spent installing/upgrading
- Total: 392 (529MB with xz compression)
- Current count without -debug/-development/-profile: 118 (158MB with xz compression)
- Current count without -debug/-development/-profile/-lib32: 80 (150MB with xz compression)
- Number of packages will only go up starting now
WITH_/WITHOUT_ interaction

- WITH_ and WITHOUT_ control what we build (see src.conf(7))
WITH_/WITHOUT_ interaction

- WITH_ and WITHOUT_ control what we build (see src.conf(7))
- Some simply exclude one component from the system (WITHOUT_APM or WITHOUT_AMD)
WITH_/WITHOUT_ interaction

- WITH_ and WITHOUT_ control what we build (see src.conf(7))
- Some simply exclude one component from the system (WITHOUT_APM or WITHOUT_AMD)
- Some change binaries (WITHOUT_KERBEROS or WITHOUT_CAPSICUM)
WITH_/WITHOUT_ interaction

- WITH_ and WITHOUT_ control what we build (see src.conf(7))
- Some simply exclude one component from the system (WITHOUT_APM or WITHOUT_AMD)
- Some change binaries (WITHOUT_KERBEROS or WITHOUT_CAPSICUM)
- No real solution for this now, we would need flavors like in ports
How to bootstrap packages (the correct way)
How to bootstrap packages (the correct way)

- Define WITH_REPRODUCIBLE_BUILD in src.conf
How to bootstrap packages (the correct way)

- Define WITH_REPRODUCIBLE_BUILD in src.conf
- Define SOURCE_DATE_EPOCH in env
How to bootstrap packages (the correct way)

- Define WITH_REPRODUCIBLE_BUILD in src.conf
- Define SOURCE_DATE_EPOCH in env
- Pass REPODIR to make(1)
How to bootstrap packages (the correct way)

- Define `WITH_REPRODUCIBLE_BUILD` in `src.conf`
- Define `SOURCE_DATE_EPOCH` in `env`
- Pass `REPODIR` to `make(1)`
- `make packages`
How to generate packages for -p updates

- Define WITH_REPRODUCIBLE_BUILD in src.conf
- Define SOURCE_DATE_EPOCH in env (same one as the bootstrap one)
- Pass PKG_VERSION to make(1) (same value as the bootstrap one)
- Use a temporary REPODIR
- Compare packages with the bootstrap ones
- Regenerate package with new SOURCE_DATE_EPOCH and new PKG_VERSION
- rm packages from original repo, copy new ones and re-run pkg repo
- User now only have to download/install package(s) affected by the SA/EN
How to generate packages for -p updates

- Define WITH_REPRODUCIBLE_BUILD in src.conf
- Define SOURCE_DATE_EPOCH in env (same one as the bootstrap one)
- Pass PKG_VERSION to make(1) (same value as the bootstrap one)
- Use a temporary REPODIR
- Compare packages with the bootstrap ones
- Regenerate package with new SOURCE_DATE_EPOCH and new PKG_VERSION
- rm packages from original repo, copy new ones and re-run pkg repo
- User now only have to download/install package(s) affected by the SA/EN
How to generate packages for -p updates

- Define `WITH_REPRODUCIBLE_BUILD` in `src.conf`
- Define `SOURCE_DATE_EPOCH` in `env` (same one as the bootstrap one)

- Pass `PKG_VERSION` to `make(1)` (same value as the bootstrap one)
- Use a temporary `REPODIR`
- Compare packages with the bootstrap ones
- Regenerate package with new `SOURCE_DATE_EPOCH` and new `PKG_VERSION`
- `rm` packages from original repo, copy new ones and re-run `pkg repo`
- User now only have to download/install package(s) affected by the SA/EN
How to generate packages for -p updates

- Define WITH_REPRODUCIBLE_BUILD in src.conf
- Define SOURCE_DATE_EPOCH in env (same one as the bootstrap one)
- Pass PKG_VERSION to make(1) (same value as the bootstrap one)
- Use a temporary REPODIR
- Compare packages with the bootstrap ones
- Regenerate package with new SOURCE_DATE_EPOCH and new PKG_VERSION
- rm packages from original repo, copy new ones and re-run pkg repo
- User now only have to download/install package(s) affected by the SA/EN
How to generate packages for -p updates

▶ Define WITH_REPRODUCIBLE_BUILD in src.conf
▶ Define SOURCE_DATE_EPOCH in env (same one as the bootstrap one)
▶ Pass PKG_VERSION to make(1) (same value as the bootstrap one)
▶ Use a temporary REPODIR
How to generate packages for -p updates

- Define WITH_REPRODUCIBLE_BUILD in src.conf
- Define SOURCE_DATE_EPOCH in env (same one as the bootstrap one)
- Pass PKG_VERSION to make(1) (same value as the bootstrap one)
- Use a temporary REPODIR
- Compare packages with the bootstrap ones

rm packages from original repo, copy new ones and re-run pkg repo

User now only have to download/install package(s) affected by the SA/EN
How to generate packages for -p updates

- Define WITH_REPRODUCIBLE_BUILD in src.conf
- Define SOURCE_DATE_EPOCH in env (same one as the bootstrap one)
- Pass PKG_VERSION to make(1) (same value as the bootstrap one)
- Use a temporary REPODIR
- Compare packages with the bootstrap ones
- Regenerate package with new SOURCE_DATE_EPOCH and new PKG_VERSION

rm packages from original repo, copy new ones and re-run pkg repo

User now only have to download/install package(s) affected by the SA/EN
How to generate packages for -p updates

- Define WITH_REPRODUCIBLE_BUILD in src.conf
- Define SOURCE_DATE_EPOCH in env (same one as the bootstrap one)
- Pass PKG_VERSION to make(1) (same value as the bootstrap one)
- Use a temporary REPODIR
- Compare packages with the bootstrap ones
- Regenerate package with new SOURCE_DATE_EPOCH and new PKG_VERSION
- rm packages from original repo, copy new ones and re-run pkg repo
How to generate packages for -p updates

▶ Define WITH_REPRODUCIBLE_BUILD in src.conf
▶ Define SOURCE_DATE_EPOCH in env (same one as the bootstrap one)
▶ Pass PKG_VERSION to make(1) (same value as the bootstrap one)
▶ Use a temporary REPODIR
▶ Compare packages with the bootstrap ones
▶ Regenerate package with new SOURCE_DATE_EPOCH and new PKG_VERSION
▶ rm packages from original repo, copy new ones and re-run pkg repo
▶ User now only have to download/install package(s) affected by the SA/EN
pkg groups

- Meta-pkg at the repo level
pkg groups

- Meta-pkg at the repo level
- Installer install "FreeBSD-base" "FreeBSD-debug" "FreeBSD-lib32" etc ...
pkg groups

- Meta-pkg at the repo level
- Installer install "FreeBSD-base" "FreeBSD-debug" "FreeBSD-lib32" etc ...
- New packages are installed automatically on update
pkg groups

- Meta-pkg at the repo level
- Installer install “FreeBSD-base” “FreeBSD-debug” “FreeBSD-lib32” etc ...
- New packages are installed automatically on update
- Multiple candidates for one package (-noman, -nocapsicum)
Current work

- bsdinstall support
Current work

- bsdinstall support
- release image support
Current work

- bsdinstall support
- release image support
- kernel-select
Current work

- bsdinstall support
- release image support
- kernel-select
- more packages split
Future work

- Talk to re@ so we have official packages
Future work

- Talk to re@ so we have official packages
- “freebsd-update”
Future work

- Talk to re@ so we have official packages
- "freebsd-update"
- poudriere image support
Are we there yet?

Not yet but close

Wanna help?

Still a few bugs in bsd.*.mk

Test installing a minimal FreeBSD based pkgbase and install each package separately to test if everything is working

pkgbase@freebsd.org
Are we there yet?

- Not yet but close
Are we there yet?

- Not yet but close
- Wanna help?
- Still a few bugs in bsd.*.mk
- Test installing a minimal FreeBSD based pkgbase and install each package separately to test if everything is working
- pkgbase@freebsd.org
Thanks

- Baptiste Daroussin (bapt@FreeBSD.Org)
- Glen Barber (gjb@FreeBSD.Org)
Questions ?
Emmanuel Vadot
manu@freebsd.org
Twitter: @manuvadot