NetBSD 基本システムのパッケージ化の設計と実装

The Design and Implementation of the NetBSD Base System Packaging

千年科学技術大学 大学院
光科学研究科 光科学専攻
博士前期課程

主査　小松川 浩　教授
副査　吉田 淳一　教授
副査　不破 泰　　教授
副査　深町 賢一　専任講師

M2160020
榎本 優樹

目次

凡例

本稿ではプログラムの名称は `ls` のようにタイプライタ調で記載し，ディレクトリやファイル名は **/etc/passwd** のように太文字で記載する．現行版のシステムについて言及している場合は，`ls` (1) のようにファイル名やプログラム名に “(数字)” を付け，オンラインマニュアルの章番号を明記する．

コマンドライン操作とその結果を示すさいに

```
$ echo "test"
test
$
```

と表記する場合もある．ここで，行頭の`$`は一般ユーザにおけるコマンドライン操作を表している．一方，root ユーザのコマンドライン操作は行頭に`#`を用いる．

UNIX®は The Open Group の商標である．本稿で UNIX®および UNIX®から派生した BSD，MINIX，Linux などのオペレーティングシステム (OS) について広く言及する場合，UNIX®から商標記号を省略し，大文字による綴りをせず Unix と表記する．

本稿では Free Software や Open Source Software のようなソースコードを公開し開発がおこなわれているソフトウェアを指す場合，Free Software と Open Source Software を足し合わせた “Free Libre Open Source Software” の略である “FLOSS” という語を用いる．

1 序論

1.1 Unix オペレーティングシステム

Unix OS は商用・非商用を問わず開発が続けられ，世界中のサービスやサーバで運用されている．商用 Unix の例としては Oracle 社の Solaris，Hewlett-Packard 社の HP-UX，Red Hat 社の Red Hat Enterprise Linux が挙げらる．非商用 Unix としては Debian や CentOS に代表される Linux ディストリビューションや，FreeBSD や NetBSD に代表される BSD Unix が挙げられる．

NET MARKETSHARE によるデスクトップおよびラップトップ OS の市場シェア調査によると，Unix ではない Windows 系 OS が 80% 以上のシェアを占めている．しかしスマートフォンでは Linux カーネルが使われている Android が，2017 年 11 月の時点では最も高いシェアを占めている [1]．またそれに次ぐ iOS も Darwin と呼ばれる，Apple 社によって開発されている，Unix と互換性があるカーネルが使われている．

1.2 Unix におけるセキュリティと運用

Unix は世界中の開発者によって機能の拡張がおこなわれている．その一方でサービス運用者はインストールされているプログラムの脆弱性やシステム設定の不備を確認・修正し，サービスとして運用されている Web サーバまたは IoT 機器を対象としたクラッキングに対して常に対策を講じ続けなければならない．

トレンドマイクロによれば，2016 年上半期は Bash の Shellshock や Bash Bug による脆弱性を利用した攻撃が増加したとの報告があげられている [2]．また Linux を搭載している IoT 機器を対象とした MIRAI や IMEIJ といった DDoS 攻撃用不正プログラムの発見が報告されており [3]，Unix におけるセキュリティアップデートの重要性はもはや無視できないものとなっている．

アップデートを確認する対象は Bash のようなユーザランド内のプログラムに限らず，カーネルに対してもおこなわなければならない．図 1 は 1999 年から 2017 年にかけて報告された Linux カーネルの脆弱性報告の数である [4]．

あらゆるバージョンに対する脆弱性報告の数を数えているが，特に 2017 年は脆弱性報告数が最多となった．このように，運用保守をおこなう者はカーネル・ユーザプログラム双方のセキュリティアップデートについて，常に注視していなければならない．

1.3 Unix，特に BSD Unix における運用上の問題点

このようにアップデート作業をおこなわなければならない状況は多く考えられ，迅速なアップデートを提供するシステムはセキュリティ面において重要な役割を果たす．

一般に，セキュリティアップデートの際は開発プロジェクトから提供された修正パッチをソースコードに適用し，ソースコードをビルドしてシステムにインストールし直す．しかしこの方法はビルド時間が CPU やメモリ資源に左右されてしまい，システムによって修正速度に差が出てしまう．そこで Unix ではコンパイル済みのプログラムとライブラリ・設定ファイル・ドキュメントに加え，管理用プログラムが解釈可能なメタ情報を単一のアーカイブに格納した「パッケージ」と呼ばれるファイルを扱うことによって，修正パッチの適用やソースコードのビルドが省略可能となっている．Linux ディストリビューションや BSD ファミリでは，パッケージはインストーラが提供しないソフトウェアである第三者製ソフトウェアの管理に用いられるという点で

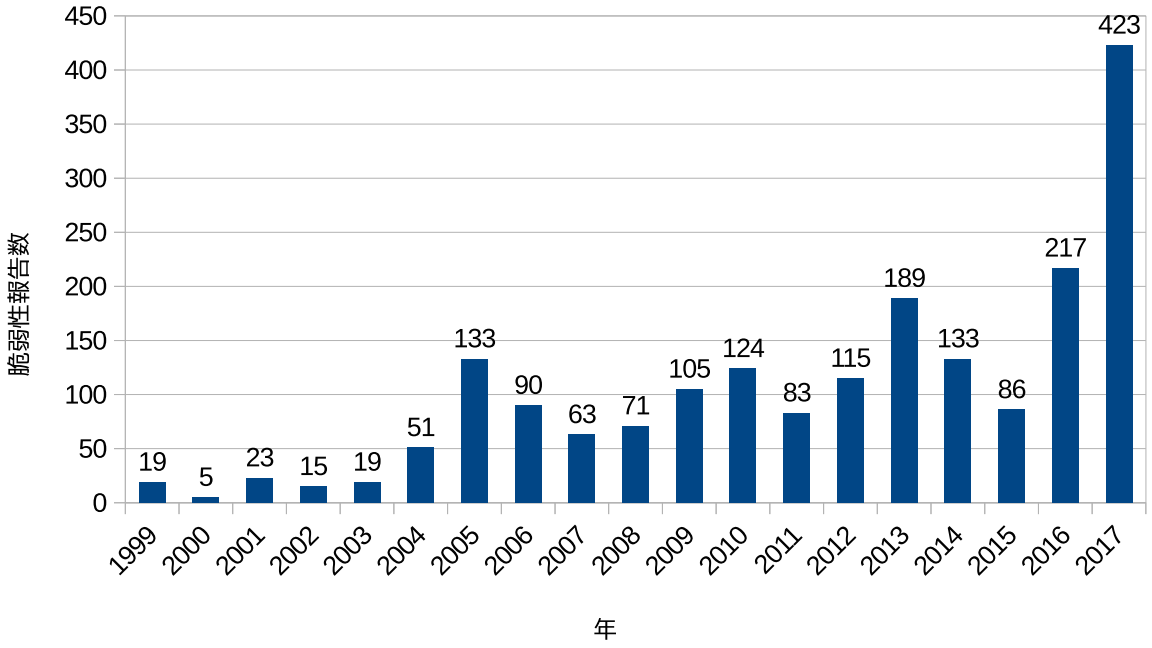


図 1　Linux カーネルの脆弱性報告の数

共通している．

しかしながら基本システムの管理方式は Linux と BSD では大きく異なり、Debian や Red Hat Enterprise Linux といった Linux ディストリビューションでは，OS の構成要素をすべてパッケージで管理することでシステム管理を容易にしている．これは開発体制以前に、歴史的経緯に由来すると考えられる。

Linus Torvalds は Linux カーネルのみを開発した．したがってユーザプログラムはすべて外部のものを利用する必要があった．Linux カーネルを使ったシステムを配布する場合，第三者製のソフトウェアをなんらかの方法で管理する必要があった．結果，Debian プロジェクトや Red Hat 社が選んだように，システムの構成要素をパッケージとして管理する手法がディストリビューションを作成する上で都合がよかったと考えられる．

一方，BSD Unix は AT&T 社の UNIX を改良し，テープとして配布していた歴史を持つ．このテープにはカーネルやユーザプログラムなどシステムに必要なものはすべて同梱されていた．同梱されていたプログラムのソースコードはすべて単一のソースツリー内で管理されていた．この単一のソースツリーに収録されているものすべてを「基本システム」と呼ぶ．

BSD Unix のリリースはこの基本システムに変更を加え続けることでおこなわれてきた．開発プロジェクトごとにカーネルやユーザプログラムを含んだ単一のソースツリーを管理しているため，基本システム全体をパッケージ管理する必要は無かった．これは Linux ディストリビューションのように，カーネル以外は第三者製ソフトウェアを集めて構成されているシステムとは異なった管理方法である．今日に至っても BSD Unix が Linux ディストリビューションのように基本システムがパッケージ管理化されていない原因は，このようなシステムの開発方式の違いによるものだと言える．

今日では BSD Unix は多く存在する．特に FreeBSD と NetBSD に注目すると，基本システムをパッケー

ジ化するシステムについて，NetBSD は 2002 年に syspkg が基本システムに取り込まれた．また FreeBSD は FreeBSD-11.0 から PkgBase が基本システムに取り込まれた．しかし syspkg は開発が停滞し未だ完成に至っていない．PkgBase はベータ版として扱われており，Debian や Red Hat Enterprise Linux などに遅れをとっている状況である．

1.4 本稿の構成

OS の構成は歴史的な事情から Linux ディストリビューションと BSD Unix に違いがみられる．しかし現代におけるシステム運用を考慮すると，BSD Unix のようなソースツリーによる基本システムの管理より，Linux ディストリビューションのような，パッケージによるシステム全体の管理が速度面また機能面から見ると望ましい．

Linux ディストリビューションは BSD Unix より優れたシステム管理の方式を採っているが，商用利用を検討した場合，Linux カーネルや Linux ディストリビューションのユーザプログラムに多く採用されているライセンスは伝播性のある GNU GPL であり，改変したソースコードを公開をしなければ訴訟のリスクが考えられる．一方，BSD Unix に採用されているライセンスはソースコードの公開義務がないため，商用製品に適している．

本研究では NetBSD に注目し，セキュリティアップデートの重要性と NetBSD の利点を踏まえた上で，syspkg の代替となるプログラムである basepkg を開発した．basepkg は syspkg のように NetBSD の基本システムをパッケージ化する．また機能面において syspkg 以上の機能の実装をおこなうことができた．また NetBSD Project 非公式ではあるが，basepkg によるパッケージ配布サーバを運用し，NetBSD 基本システムのパッケージ管理サービスを提供している．

本稿では Unix と Free Libre Open Source Software (FLOSS) の歴史的背景について年代別に説明したのちに，Unix の構成要素および FLOSS の代表的なライセンスである GNU GPL と BSD License について述べる．その後，basepkg の設計と実装を述べ，basepkg の開発と運用を通じて得られた手法やデータから，syspkg と比べ NetBSD の基本システムのパッケージ化がどのように改善されたかを述べる．

2　1960 年から 1990 年代の Unix およびソースコード共有文化の歴史的背景

FreeBSD や NetBSD のような BSD 系 Unix と，Debian や Red Hat Enterprise Linux のような Linux ディストリビューションの間には，システム構成や開発体制およびライセンスの違いが見受けられる．特にシステム構成やライセンスの違いを理解するには，これら Unix が誕生するに至った歴史的背景や，プログラマの文化的背景を抑えなければならない．したがって本章では 1960 年から 1990 年代にかけての Unix の開発や，プログラムのソースコードを共有する文化の成り立ちについて時系列順に述べる．

2.1　Tech Model Railroad Club

今日の FLOSS 思想の基礎を作り上げたのは Massachusetts Institute of Technology (MIT) の Tech Model Railroad Club (TMRC) である．この同好会は主に列車のレプリカやジオラマを作る「ナイフと絵筆」派と，実際に模型を動かすシステムを作る「信号と動力」派に分かれていた [5, pp. 9]．プログラミング分野においては後者に所属しているプログラマが注目を集め，本稿で TMRC に触れる際は後者を指すこととする．

MIT に TX-0 コンピュータが導入され，TMRC のメンバがこのコンピュータでプログラミングをおこなうようになってから「コンピュータへのアクセスを制限してはならない」「情報はすべて自由に利用できなければならない」といった “Hacker Ethic” と呼ばれる文化が形成されていった [5, pp. 32-34]．ここで “Hacker” とは「プログラム可能なシステムの細かい部分を探ったり、その機能を拡張する方法を探求したりするのに喜びを感じる人。」[6, pp. 254] を指す．メディアでは「サイバー攻撃をおこなう者」として Hacker が用いられるが，これは誤りであり，本来は “Cracker” がそれに該当する語である．

Hacker Ethic を象徴するソフトウェアとして Steve Russell，Alan Kotok，Peter Samson，Dan Edwards らによる Space War!がある．このソフトウェアは 1962 年 5 月に一般公開された．Space War!の紙テープは無料で流通し，Degital Equipment Corporation (DEC) 社の PDP-1 最終診断プログラムとして利用された．

ここで重要な点は，Space War!の利用を制限しなかったことで TMRC のメンバ誰もが自由にプレイでき，また自由に機能を追加できたことである．また TMRC 外部に紙テープが無料で流通した点も，経済的な利益よりも自由な情報の利用を優先させた事例であると言える．これは今日の FLOSS の開発モデルに類似しており，この文化の起源を 1960 年代にまで遡れると考察できる証左である．

2.2　Project MAC

1963 年，MIT は Advanced Research Projects Agency (ARPA) から，Time Sharing System の研究プロジェクトである Project MAC (Machine-Aided Cognition または Multi-Access Computer の略) のために 300 万ドルの援助を受けた．

Time Sharing System とは，カーネルが処理するプロセスを随時切り替えることで，ひとつの CPU 上で複数のプログラムを並行して処理するシステムである．このシステムにより，ひとつのコンピュータで複数のユーザがプログラムを同時に実行できる [7]．Time Sharing System は International Business Machines (IBM) の OS/360 に実装されていた Multi Programming を前身とし，Fernando J. Corbató によって Compatible Time Sharing System (CTSS) に実装された．

1965 年，Project MAC では MIT と Bell Telephone Laboratories (BTL)，General Electric (GE) 社の共同プロジェクトとして Multiplexed Information and Computing Service (MULTICS) が開発された [8]．

これは「1 つの巨大なシステムでボストン中のすべての人々に必要な計算機能力を提供することを想定していた」[9, pp. 13] システムであり，IBM の PL/I 言語で書かれた．BTL は 1969 年 3 月に Project MAC から撤退したが，その年の 10 月には MIT で MULTICS の運用が開始された [10, pp. 20-21]．また 1970 年に GE はコンピュータ事業から撤退し，Howeywell に事業を買収されることとなった．結果，MULTICS は 1973 年 1 月から Honeywell 6180 の OS として販売されることとなった [10, pp. 21]．

一方，Marvin Minsky は 1959 年に設立された Artificial Intelligence Laboratory (AI Lab) を Project MAC に引き入れた．Multics 開発とは直接の関係がない AI Lab を Project MAC に参加させられた理由を藤田は「ARPA/IPTO のミッションと整合のとりにくいタイムシェアリング開発の隠れ蓑のために人工知能研究を活用することにし、その分の予算を上積みした」[10, pp. 19] と考察している．なお，IPTO とは Information Processing Techniques Office の略であり，軍事的な指揮と統制に関する研究支援をおこなう ARPA の組織のひとつである．以降 AI Lab では TMRC の流れを汲むハッカー倫理が成熟され，人工知能研究の傍ら Richard Greenblatt のチェスソフトウェア Mac Hack，Joseph Weizenbaum の対話ソフトウェア ELAIZA が開発された．また CTSS や MULTICS に不満を抱いた Richard Greenblatt，Tom Knight，Stewart Nelson らにより Incompatible Time Sharing System (ITS) が開発された．ITS はログイン時にパスワードを使わず，あらゆるユーザのファイルにアクセス可能な点で CTSS や MULTICS とは異なっていた．

2.3 PDP-7 Unix

MULTICS の開発に参加していた BTL の Ken Thompson は，天文シミュレーションプログラム Space Travel を動かしていたコンピュータ GE 645 の性能に不満を抱き，1969 年に Space Travel を DEC の 18bit マシン PDP-7 へ移植する作業を開始した．この際，Thompson は Dennis Ritchie，Rudd Canaday らとともに “chalk file system” と呼ばれるファイルシステムを設計し，PDP-7 に実装した．

Thompson らはファイルシステムの他にコマンドインタプリタやエディタといったファイル操作用のユーティリティをアセンブリ言語で開発し，PDP-7 上で動かした．このシステムが Unix 第一版の前身となる PDP-7 Unix である．

2.4 Unix 第一版から第三版．C 言語とパイプの登場

Thompson の上司であった Joe Ossanna は PDP-7 Unix に文書処理システムとしての研究テーマを与えることで，MULTICS の失敗により OS 開発そのものを懸念する BTL 上層部への口実とした．1970 年に DEC 社の 16bit マシン PDP-11 が購入されると，Thompson は PDP-11 向けに PDP-7 Unix を書き直し，テキストエディタ `ed` を書いた．また Doug McIlroy によって組版プログラム `roff` が書かれ，文書処理システムに必要な機能が開発された．

Unix 第一版は 1971 年 11 月 3 日に，文書処理システムを求めていた BTL の特許部門へ導入された．当時の Unix は，Libes らによると「PDP-11/20 はメモリ保護機能なしで Unix を実行しており、0.5MB のディスクを備えていた。また、編集や文書整形を行う 3 人のユーザが同時にサポートされ、それと並行して Ken のグループが Unix の開発を続行していた。」[11, pp. 6] とされている．

1973 年 2 月にリリースされた Unix 第三版では Ritchie が開発した C 言語コンパイラおよび McIlroy が考案し Thompson が実装したパイプが利用可能になった．付録 A は McIlroy によるパイプのアイデアのメモをスキャンしたものである．

C 言語は B 言語を参考に開発された．B 言語は Thompson が 1970 年に開発した言語であり，B 言語は Martin Richards が開発した Basic Combined Programming Language (BCPL) に由来している．C 言語はこれらの言語と異なり，int や float といったデータ型と，構造体・共用体，グローバル変数が実装されていた [12].

パイプは

```
$ ls | wc -l
      11
$
```

のように，プログラムの出力を別のプログラムの入力に切り替える機能である．この例では `ls` (1) の標準出力を `wc` (1) の標準入力に流すことで，ディレクトリ内のファイル数を数えている．パイプが実装されたことにより「ひとつの機能だけを実行するプログラムを書く」「他のプログラムと一緒に動作するプログラムを書く」「テキストストリームを扱うプログラムを書く」Unix 哲学と呼ばれるシステム開発論が形成された [13][14].

2.5 Association for Computing Machinery (ACM)

1973 年 10 月，Thompson と Ritchie は Unix 第三版の論文を ACM Symposium on Operating System Principles (SOSP) で発表した．1973 年 11 月に Unix 第四版がリリースされ，カーネルとシェルが C 言語で書き直された．1974 年に *Communications of the ACM* に論文が掲載され，BTL 内部でのみ導入されていた Unix を教育機関が求めるようになった．これには「メーカーごとに互換性のないクローズドシステムが一般的であった当時、コンピュータを利用するにはコンピュータメーカーが提供する高額な OS や開発ツールを使わざるを得ず、それらはいずれもソースコードが非公開だった」[10, pp. 73] 背景があると藤田は考察している．

2.6 反トラスト法とユーザ互助会

1974 年 6 月の Unix 第五版は初めて大学が Unix を教育目的として公式に入手可能になった Unix であった．そして Unix は BTL が公式にサポートとバグ修正をおこなわないという条件で配布された．

この配布条件は，1949 年に Truman 政権が米国司法省反トラスト部を通じ，電話設備の製造の独占に対して Western Electric (WE) 社と American Telephone and Telegraph (AT&T) 社を提訴し，1956 年 1 月 24 日の同意審決により WE 社と AT&T 社が敗訴した事例が背景となっている．この同意審決により AT&T 社は他企業に対する特許の公開と情報提供が義務付けられ，業務範囲を公衆通信サービスに限定された [13][15]. 特に，特許は発明の保護を目的とした制度であるものの，AT&T 社らによる独占を防ぐため発明の実用化を制限させず，AT&T 社らは「名目的な対価で、誰にでもライセンスを供与しなければならない」[13, pp. 68] 条項が同意審決に加えられていた．

WE 社は AT&T 社の子会社であり，BTL は WE 社と AT&T 社の共同出資会社であった．そのため BTL はこの同意審決に違反しないよう Unix をライセンスし，外部へ配布する必要があった．

初期の Unix ライセンス文書では非独占かつ譲渡不可能であることを条件として，Unix の使用を使用料なしで大学や教育機関に許可していた [16, 2.0.1]．また被許諾団体の従業員や学生は，必要であれば誰でも Unix のソースコードを利用できた [16, 4.0.5]．このライセンス文書のコピー全文を付録 B として添付した．

BTL からの公式なサポートを受けられなかった影響により，Unix ユーザは MIT AI Lab 発祥のプログラム開発文化のようにプログラムやバグ修正の情報を共有し，互いに協力することで開発や問題解決をおこなった．

1977 年，John Lions は University of New South Wales の講義資料として，Unix 第六版の全カーネルソースコードと注釈・アルゴリズムの解説を載せた *Source Code and Commentary on UNIX level 6* を書いた．この本は Unix のライセンスを保持していれば学生に限らず誰でも入手可能であった [17].

Mel Ferentz は “UNIX NEWS” という名の刊行物を発行し Unix ユーザへ送っていたが，商標の都合により 1977 年 7 月に “;login:” という名称に変更した．1978 年，;login:の発行手数料を管理するため “USENIX” と呼ばれる委員会が結成され，ユーザ会議の主催やソフトウェアの配布もおこなうようになった [13][11].

2.7 Berkeley Software Distribution (BSD)

1974 年 1 月，Robert S. Fabry と Keith Standiford によって University of California, Berkeley (UCB) へ Unix が導入された．1975 年に Thompson が客員教授として UCB に就任した．この際に Thompson によってバグ修正をテープに集めた “50 Bugs” が非公式に配布された．

1976 年に Queen Mary University of London の George Coulouris が開発した `em` エディタをもとに，UCB の大学院生 Chuck Haley と Bill Joy によって `ex` ラインエディタが開発された．また 1977 年に，二人は改良された Pascal コンパイラを完成させた．

「UNIX Pascal システムのうわさは UCB の外部にもすぐに伝わり、配布を求める声が UCB に集まる」[10, pp. 121] ようになったことで，1978 年，Joy によって Pascal システムと `ex` が同梱され，50 Bugs の修正パッチが当てられた 1BSD が 50 ドルで配布されるようになった．同年，スクリーンエディタ `vi` と，端末情報のデータベースである `termcap` が同梱されている 2BSD が配布された [13].

2.8 Unix 第七版とライセンス改訂

1979 年 1 月，Unix 第七版がリリースされた．この版では Alfred V. Aho・Peter J. Weinberger および Kernighan によるプログラミング言語 `awk`，Stephen C. Johnson による C ソースコード検証プログラム `lint`，Stuart I. Feldman によるプログラム管理用プログラム `make`, Mike Lesk による `uucp`，Stephen R. Bourne による `sh`，Dick Haight による `find`，`cpio`，`expr` が新たにインストールされた．

一方で第七版から Unix のライセンスが改訂された．この改訂では「授業でソースコードを学習することを禁止する旨，ライセンスに明記して，企業機密としての価値を守ろうとした」[18, pp. 15]．この改訂の影響で Lions の講義資料は発行不可となった．またこれに反発した Andrew S. Tanenbaum によって，1987 年に MINIX [18] が開発されることとなる．

2.9 Unix/V32 と 3BSD

Unix は PDP-11 のような 16bit マシンで動作する OS であったが，BTL の John Reiser と Tom London により，DEC 社の 32bit マシンである VAX-11/780 へ Unix 第七版が移植された．移植版の Unix は Unix/32V と呼ばれた．VAX-11/780 は Memory Management Unit (MMU) を備えていたものの，32V ではサポートされていなかった．この版をもとに UCB の Özalp Babaoğlu と Joy によって仮想メモリシステム・ページ置換アルゴリズムが実装され，1979 年に Bill Joy が 3BSD として配布した [19][20].

2.10　1980 年代の AT&T 社

1974 年の司法省による反トラスト法訴訟は，1982 年 1 月 8 日に 1956 年の同意審決を修正する合意を司法省と AT&T 社が取り，1982 年 8 月にワシントン DC 連邦地方裁判所が受諾した．これにより 1956 年の同意審決が修正された．修正同意審決の内容は以下の通りである．総務省の昭和 57 年版通信白書から引用・編集した．

1. AT&T は，22 のベル系電話運用会社 (BOC's) を AT&T から分離する．
2. AT&T は，長距離回線部門，製造部門の WE 及び研究開発部門の BTL を引き続き所有する．
3. AT&T は，データ処理・通信分野への参入及び宅内機器の販売を許される．
4. BOC's は，AT&T の長距離回線部門だけでなく，すべての長距離通信事業者に対し同じ条件で BOC's の回線を利用させる．
5. BOC's は，電話機及び事務所用交換台の提供等，収益の見込まれる市場に参入できる．
6. BOC's は，多大な広告収入が得られる職業別電話帳 (イエロー・ページ) を発行できる．
7. AT&T は，自社の伝送施設を用いニュース，生活情報，広告宣伝の収集，編集，配布等のエレクトニック・パブリッシングの提供を今後少なくとも 7 年間禁止される．
8. ベル・システムの組織再編成は，AT&T と司法省によってのみ実施されるものではなく，BOC's の財政的な基盤を確立させる意味からも分離する各段階で再吟味，承認が必要であり，裁判所が監督する．[21]

同年，AT&T 社内の Unix Support Group (USG) が保守していた Unix をもとに Unix System Ⅲ がリリースされた．

1983 年には USG と，AT&T 社内の Business Information Systems Programs 部門の Programmer's Workbench (PWB) が組織改編により統合し Unix System Development Laboratory (USDL) が設立された．同年，USDL によって Unix System V がリリースされた．当時のライセンス料金は System Ⅲ，System V，第七版一律で 400 ドルであった [22].

修正同意審決によりコンピュータ業界へ参入可能になった AT&T 社は 1984 年，Unix System V Release 2(SVR2) をリリースし，ライセンス料金を引き上げた．1983 年のライセンス料金に比べ，ひとつの CPU につきひとつの Unix ライセンスを購入しなければならず，BASIC インタプリタや文書整形プログラムも同様にひとつの CPU ごとにライセンスを購入しなければならなくなった [22].

1983 年の Unix ライセンス料金表と 1984 年の Unix ライセンス料金表をそれぞれ付録 C と付録 D に添付した．

2.11　4BSD

本節では 1980 年代の UCB における BSD のリリースについて順を追って述べる．

1980 年代初頭，一般的な OS への TCP/IP の実装を目的として，ARPA と Defense Communications Agency (DCA) は研究者に対する補助金の交付を決定した [23]．ここで，OS はページングシステムを備えた VAX-11 上で動作する，UCB の BSD が選ばれた．UCB の Bob Fabry はこの補助金をもとに BSD の開発グループである Computer Systems Research Group (CSRG) を設立した．

1980 年 10 月，CSRG は VAX-11/750 をサポートする 4BSD をリリースした．Joy は David Kashtan による 4BSD の性能に対する指摘に反証する論文 *Comments on the performance of UNIX on the VAX* を書き，1981 年 6 月にこの論文の成果をもとに 4.1BSD がリリースされた [24][25].

1983 年 8 月，4.1aBSD・4.1bBSD・4.1cBSD とマイナーアップデートを経て 4.2BSD がリリースされた．TCP/IP が実装され，遠隔操作用プログラム `rcp`・`rsh`・`rlogin`・`rwho` などが加わった [24]．またこの版から Fast File System (FFS) が実装された．

1986 年 6 月には 4.2BSD の性能を改善した 4.3BSD がリリースされた．また VAX 以外のアーキテクチャをサポートするためにカーネルの機種依存部分 (MD) と非依存部分 (MI) を分離する作業がおこなわれ，Computer Consoles Incorporated (CCI) 社の Power 6/32 で動作可能な 4.3-Tahoe が 1988 年 6 月にリリースされた．CSRG の Marshall K. McKusick は「BSD はその後も様々なアーキテクチャのマシン上に移植されることになったので、カーネルのコードを機種依存の部分とそうでない部分に分けたことは結果的に非常に有意義なことだった。」[24, pp. 79-81] と述べており，後の NetBSD のような，カーネルの MD・MI 部分を切り分ける設計思想の先駆けであったと考えられる．

1989 年 6 月，CSRG は TCP/IP ネットワーク機器を販売するベンダからの要望を受け，BTL の Unix/V32 のソースコードとは独立した TCP/IP 実装部分とそのユーティリティを anonymous FTP サーバ上で公開した．これは Networking Release 1(NR1) と呼ばれた．公開されたソースコードは，UCB の著作権情報を残し，帰属を明記すれば自由に利用・再頒布が可能であった．これは現在の BSD License の元祖であったと考えられる．

2.12 GNU Project と Free Software 運動

1980 年代初頭にさかのぼるが，MIT の AI Lab では，プログラム言語 LISP の処理に特化した LISP マシンと呼ばれるコンピュータの商業化に関する，Greenblatt と Russell Noftsker・Knight らの対立が生まれていた．Greenblatt の Lisp Machine 社派と Nofsker・Knight らの Symbolics 社派に商業展開が二分し，AI Lab の人員が Symbolics 社に取り込まれていった [5]．結果，「人工知能研究所は人手不足に陥り、自立したコミュニティとして機能できなくなってしまった」[26, pp. 105]．Symbolics 社の LISP マシンは MIT に販売される契約であったが，開発されたシステムはすべて企業秘密であった．

Symbolics 社の人材の引き抜きによって，AI Lab では DEC の PDP-10 上で動作していた ITS を自力で保守できなくなり，代わりに有料かつソースコードが非公開の DEC 社製タイムシェアリングシステムが PDP-10 へ導入された．AI Lab における Hacker 文化の崩壊をうけ，1984 年 1 月，AI Lab のメンバであった Richard M. Stallman は MIT を退職し，GNU Project を発足させた．GNU とは “GNU's Not Unix” の再帰的な略称である．また 1985 年には GNU Project の運営を目的に Free Software Foundation (FSF) が設立された．

GNU Project の目的は，Unix と互換性があるシステムを目指し，「実行、コピー、配布、研究、変更、改良する自由を利用者が有する」[27] ソフトウェアを意味する “Free Software” の開発である．以下を満たすソフトウェアは Free Software であるとされる．以下の箇条書きは『自由ソフトウェアとは?』[27] から引用した．

- どんな目的に対しても、プログラムを望むままに実行する自由 (第零の自由)。
- プログラムがどのように動作しているか研究し、必要に応じて改造する自由 (第一の自由)。ソースコードへのアクセスは、この前提条件となります。

- 身近な人を助けられるよう、コピーを再配布する自由 (第二の自由)。
- 改変した版を他に配布する自由 (第三の自由)。これにより、変更がコミュニティ全体にとって利益となる機会を提供できます。ソースコードへのアクセスは、この前提条件となります。[27]

Free Software の開発と利用を促す運動は Free Software 運動と呼ばれた．

2.12.1 GNU General Public License

Stallman は Free Software を配布するさい，配布形態に “Copyleft” という表現を用いた．Copyleft について GNU Project は「プログラム (もしくはその他の著作物) を自由 (自由の意味において。「無償」ではなく) とし、加えてそのプログラムの改変ないし拡張されたバージョンもすべて自由であることを要求するための、一般的な手法の一つ」[28] としている．これは Public Domain のように著作権を放棄させず，かつ Free Software を自由に利用可能にする法的保護を目的とした概念である [26]．

具体的に，Copyleft は GNU General Public License (GNU GPL) の形で表れている．GNU GPL はソフトウェアライセンスの一種であり，第 1 版は 1989 年 2 月 1 日に発行された [29]．GNU GPL で保護されたソフトウェアは有償・無償に関わらずソースコードが公開され，それに対する複製・再頒布・改変は制限されてはならない．またソフトウェアとそのソースコードは無保証で提供される．加えて，GNU GPL で保護されたソースコードが使われているソフトウェアはすべて GNU GPL で保護しなければならないという Copyleft の概念が取り込まれている．

2.13 BSD ファミリの登場と 4.4BSD-Lite Release 2 まで

2.13.1 NR2 と 386BSD およびその子孫

NR1 が Unix コミュニティで好評価を得られていたことを受け，CSRG の Keith Bostic は AT&T 社の Unix のソースコードを使わず，AT&T 社から Unix を購入せずとも利用・再頒布可能な Unix の作成を提案した．1988 年 7 月 21 日，Bostic は USENET の comp.bugs.4bsd.ucb-fixes Newsgroup でソフトウェアの寄付を募った．図 2 は USENET のアーカイブから引用した Bostic によるメール全文である．

この呼びかけにより，有志らが開発した Unix コマンドおよび C ライブラリが寄付され，Unix/V32 のものを置き換えることが可能となった．これを受け，CSRG の Mike Karels と McKusick により，ソースファイル 6 つを除いて Unix/V32 とは独立したカーネルが開発された．このシステムは Networking Release 2(NR2) と呼ばれ，6 つのファイルが欠けた状態で，1991 年 6 月，NR1 と同様に anonymous FTP サーバ上で配布された．

1992 年，Bill F. Jolitz によって NR2 に欠けている部分を補い，ひとつのシステムとして動作する 386BSD がリリースされた．NR1・NR2 同様に anonymous FTP サーバで配布された．その後，386BSD をもとにした商用 Unix の開発・販売をおこなう Berkeley Software Design Incorporated (BSDI) が設立された [24]．

1993 年，386BSD のバグ修正や機能拡張を目的とし，Chris Demetriou・Theo de Raadt・Adam Glass・Charles M. Hannum によって The NetBSD Project が発足し，1993 年 4 月に NetBSD 0.8 がリリースされた [30]．同年，Nate Williams・Rod Grimes・Jordan Hubbard によって The FreeBSD Project が発足し，1993 年 12 月に FreeBSD 1.0 がリリースされた [31]．

```
From: bostic@OKEEFFE.BERKELEY.EDU (Keith Bostic)
Newsgroups: comp.bugs.4bsd.ucb-fixes
Subject: V1.62 (Request for user-contributed software)
Message-ID: <8807212226.AA02439@okeeffe.Berkeley.EDU>
Date: 21 Jul 88 22:26:23 GMT
Sender: daemon@ucbvax.BERKELEY.EDU
Organization: University of California at Berkeley
Lines: 33
Approved: ucb-fixes@okeeffe.berkeley.edu

Subject: Request for user-contributed software

BSD will be starting a new cycle of software development this summer.  As
most of you know, much of the software made available through Berkeley was
contributed by the user community.  We wish to continue this tradition and
encourage anyone who is interested in contributing software to the user
community to contact us.  We also have many projects, ranging in difficulty
from weekend hacks to master's or doctorate level work, that we do not have
time to do ourselves.

Possible projects include:

    autodialer manager/daemon
    biff(1) replacement (mail notification)
    conferencing system
    csh cleanups and the addition of ksh features, particularly
        functions and command line editing
    System V compatible cron(8) and at(1) programs
    documentation browsing system
    multi-buffered tape driver
    multiuser, message-based application scheduler
    public-domain NFS
    replacement of current various current utilities with
        public domain source code
    make(1) replacement

We would appreciate being contacted by anyone interested working on these,
or any other, projects.

Keith Bostic
415-642-4948
uunet!keith
bostic@okeeffe.berkeley.edu
```

図 2　Bostic によるソフトウェア寄付の呼びかけ全文

2.13.2 訴訟問題

AT&T 社が筆頭株主の Unix System Laboratories (USL) 社は 1992 年，BSDI を相手に Unix 製品の出荷差し止めを求める訴訟を起こした．この裁判は USL 社が敗訴した．USL 社は再審議を裁判所に申し立て，BSDI と UCB に対し訴訟を起こしたが，その再審請求は却下された．1993 年 6 月，報復として UCB は USL 社に対し，BSD のライセンス契約に関する訴訟を起こした．同年，Novell 社が USL 社を買収し，1994 年 2 月 4 日に「NR2 を構成する一万八千のファイルから三個のファイルを削除し，それ以外のファイルのいくつかに若干の変更を加える」[24, pp. 88] ことで和解が成立した．この和解をうけ，1994 年 6 月に 4.4BSD-Lite がリリースされた．1995 年 6 月には 4.4BSD-Lite Release 2 がリリースされ，CSRG は解散した [24].

一方でこの訴訟の和解により 386BSD から分岐した FreeBSD や NetBSD の開発グループは，4.4BSD の変更に追従する作業をおこなわなければならなくなった．4.4BSD のソースコードをもとに，1994 年 11 月に NetBSD 1.0 と FreeBSD 2.0 がリリースされた [32][31].

2.14 Linux とその広がり

Unix 第 7 版リリース時のライセンス改定に反発した Andrew S. Tanenbaum は，Unix のソースコードを使わずに Unix と互換性を保つ OS である “MINIX” を開発した．

1991 年，当時 University of Helsinki の学生であった Linus Torvalds は MINIX をもとに Intel/386 アーキテクチャ向けのカーネルを開発した．このカーネルは anonymous FTP サーバ上で “Linux” という名前で配布された．

Eric Raymond は Linux の開発スタイルについて「はやめにしょっちゅうリリース，任せられるものはなんでも任して，(中略) なんでもオープンにする」[33] と述べたとおり，大勢の有志らによって開発されたソースコードを可能な限り取り込み，頻繁に新版のリリースをおこなうことで，386BSD のような少人数制の開発手法とは異なる手法で機能が追加されていった．

1992 年には「Linux カーネルと，ほぼ完成状態の GNU システムとを統合し，完全な UNIX 風フリーオペレーティングシステムとして実際に使うことが可能になった」[26, pp. 131] と Stallman は述べている．事実，1992 年には Patrick Volkerding によって Slackware の開発が始まり，1993 年 7 月に初版がリリースされた [34]．これは世界初の Linux ディストリビューションとなった．1993 年 8 月には Ian Murdock によって Debian プロジェクトが開始され，Debian 0.01 がリリースされた [35].

一方で商用の Linux ディストリビューションを販売する企業も現れた．アメリカの Red Hat 社やヨーロッパの SUSE 社，日本の TurboLinux 社がその代表例である．特に Red Hat 社は 1999 年 8 月にアメリカのベンチャー向け株式市場 NASDAQ に株式公開をし，時価総額が 35 億ドルに至るほどの反響を受けた [36]．今日においても，Red Hat 社は Red Hat Enterprise Linux を企業向けに販売している．

2.15 Open Source Software

Raymond は 1997 年 5 月，Linux カーネルの開発手法とその実践に関する論文『伽藍とバザール』を発表した．Microsoft におされ業績が不調であった Netscape 社はこの論文に触発され，FLOSS コミュニティとの会議の末，1998 年 1 月，自社製品の Web ブラウザである Netscape Navigator のソースコードをすべて公開，インターネットコミュニティに自社ブラウザの将来を託した．そしてこの Web ブラウザは非営利団体

Mozilla が開発を担うようになった [37]．これが今日では広く用いられている Mozilla Firefox の起源である．なお Netscape 社は翌 1999 年 AOL 社に買収され，消滅した．

このようにフリーソフトウエア運動は多くの成果を生み出してきていたが，その一方で，Stallman の Free Software という概念は「知的所有権に反旗を翻している，共産主義的な立場をとっている」[37, pp. 407] また「リベラルなプログラマからは歓迎されていたが，ソフトウェアビジネスの人たちからは煙たがられていた」[38, pp. 334] 背景があった．そのため，企業が製品のソースコードの公開を決定し，それが商業的に受け入れられるためには FSF による Free Software 運動とは別のマーケティング活動が必要であった．

1998 年 2 月，Raymond によって “Open Source” という用語が発案され，Bruce Perens によって “Debian Free Software Guidline (DFSG)”[39] をもとに “Open Source Definition”[40] が作成された．この Open Source の定義を満たすソフトウェアは Open Source Software (OSS) と呼ばれる．Open Source という用語は商標登録され，Ryamond と Perens は Open Source Initiative を設立した．

3　Unix の技術的な概要

第 2 章では 1960 年代から 1990 年代における Unix の歴史的な背景や Linux の成り立ち，プログラマによる FLOSS 文化の形成について述べた．

本研究は FLOSS な OS である NetBSD の基本システムを扱うため，Unix の基本的な構成要素および管理方法，また FLOSS におけるライセンスについての知識が必要である．したがって本章では，Unix の特にユーザランド側の概要および FLOSS ライセンスの説明をおこなう．

3.1　Unix の構成要素 — 例．NetBSD

本節では NetBSD 7.1 を例に Unix の構成要素について述べる．

3.1.1　Unix システム構造

Unix はハードウェアやプログラムを制御する「カーネル」と呼ばれるソフトウェアと，カーネル以外の各種ファイルやディレクトリ・ユーザプログラムの集合である「ユーザランド」からなる．

Unix は図 3 のように，**root**(/) ディレクトリを頂点としてディレクトリが木構造を成している．ここで，青い楕円はディレクトリを表し，四角はファイルを表す．ファイルについて，オレンジ色の **netbsd** は NetBSD カーネルである．緑色のファイルは実行可能プログラムを表し，白は平文ファイルである．これらのファイルはすべて「ファイルシステム」と呼ばれるシステムの上で管理される．

ファイルやディレクトリの所在を表すには，絶対パスと相対パスによる表記法がある．絶対パスは木構造の頂点である/ディレクトリからみた完全なパスである．**passwd** ファイルを絶対パスで表記すると **/etc/passwd** となる．相対パスは現在のディレクトリからみたパスである．ユーザ fox が自身のホームディレクトリ **/home/fox** にいる場合，**passwd** ファイルの相対パスは**../../etc/passwd** となる．

3.1.2　ユーザ

Unix は複数のユーザが同時にログインし，プログラムを実行可能であるように設計されている．図 3 では dog と fox ふたつのユーザアカウントが存在する．ユーザ名やパスワード，本名といったユーザ情報は **/etc/master.passwd** に記録されている．ユーザは**/etc/master.passwd** に記録されたものと一致するパスワードをログイン時に入力することでシステムにログインできる．

ユーザにはユーザ固有のホームディレクトリが割り当てられており，ホームディレクトリ以下でのみファイルの読み取り・書き込み・実行すべてが可能である．したがってユーザ dog は**/home/dog** 以下では **/home/dog/src/main.sh** を編集し・実行し・削除することもできるが，ユーザ fox の **tech.txt** を編集したり削除することはできない．

3.1.3　システムの起動手順

通常，Unix システムは Hard Disk Drive(HDD) や Solid State Drive(SSD) といったデータ記録媒体に書き込まれている．図 4 は Intel 互換機に搭載された 20GB のディスクに NetBSD をインストールしたさいの，ディスクの区画を表したものである．

ディスク先頭の 512B に Master Boot Record(MBR) が書き込まれ，その次の 512B は PBR，その次の

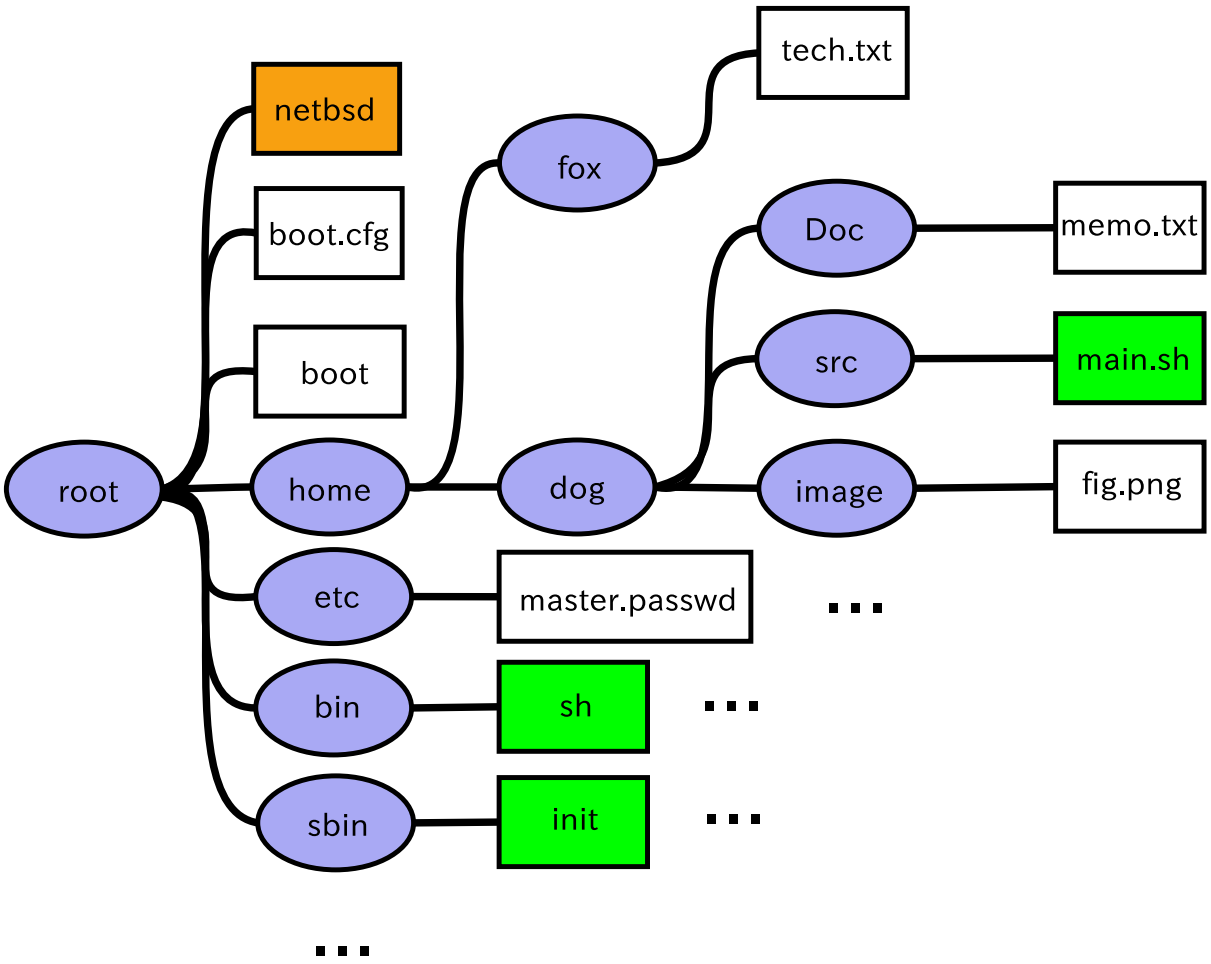


図 3　NetBSD の root を頂点としたディレクトリ構造とファイル名の例．楕円はディレクトリを，四角はファイルを表している．余白の都合上，省略しているディレクトリやファイルがある．また，このシステムにはユーザ dog と fox が存在する．

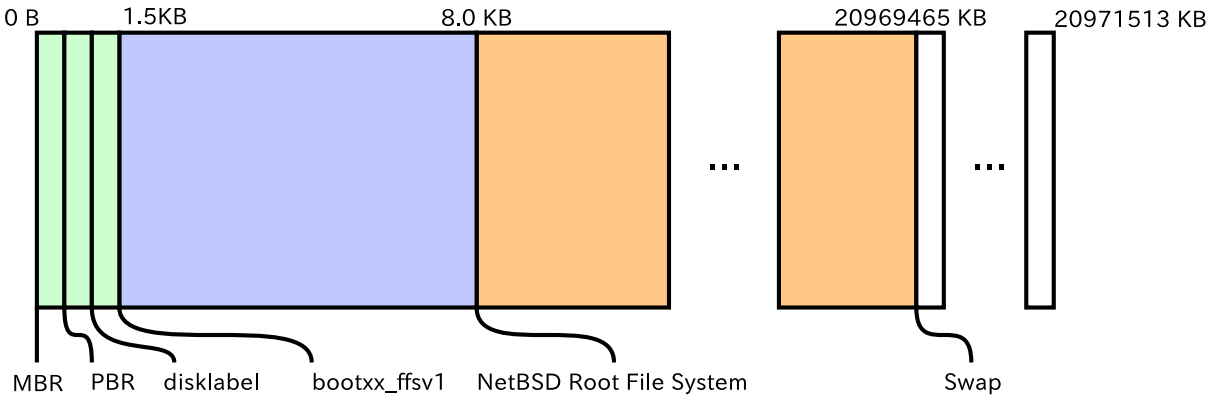


図 4　20GB のディスクに NetBSD の Root File System 18GB と Swap 領域 2GB を割り当てている．実際には MBR と bootxx_ffsv1 がディスクの先頭に書き込まれているため，ユーザが利用可能なディスクの容量は 18GB より 8KB 分少ない．

512B は disklabel，以降の 6.5KB は Partition Boot Record(PBR) が書き込まれている．これは NetBSD では bootxx_から始まるファイルとして用意されており，一般的には **bootxx_ffsv1** が使われる．以降，NetBSD のファイルシステムとスワップ領域が書き込まれる [41].

システム起動時，Basic Input Output System(BIOS) は MBR を読み込み，MBR 内のブートコードをメモリの 0x7c00 へ書き込み，そのブートコードへシステムの制御を譲渡する．ブートコードは PBR を読み取り，PBR は bootxx を実行する．bootxx は disklabel の情報をもとに NetBSD Root File System 上にある **boot** (8) を読み，それを実行する．**boot** (8) はカーネルである**/netbsd** を実行する [41].

NetBSD カーネルはシステムの初期化をおこない，問題がなければ **init** (8) を実行する．**init** (8) は **rc** (8) と呼ばれる，システムが稼働している限り実行され続ける「システムサービス」の実行を管理するプログラムを呼ぶ．次に **getty** (8) が実行され **login** (1) がユーザ名とパスワードの入力を待ち受ける．ログイン認証が通れば，シェルプログラム (標準では **sh** (1)) がユーザの入力を待ち受けるようになる．

3.1.4 インストール

Unix を HDD や SSD に書き込み，起動可能にするには，Compact Disk(CD) や Digital Versatile Disc(DVD)，Universal Serial Bus(USB) ディスクに書き込まれた「インストーラ」を用いる．インストーラはコンピュータに接続されているディスクを検出し，任意のディスクへシステムを書き込む．

NetBSD 7.1 では **sysinst** (8) によって以下の手順でインストール作業がおこなわれる [42].

1. 言語とキーボードレイアウトの設定
2. NetBSD システムを書き込むディスクの決定
3. インストールする配布物の設定
4. MBR の設定
5. ディスク全体の区画の設定
6. NetBSD ファイルシステムの区画の設定
7. インストールする配布物が収録されたメディアの設定
8. ここで実際のインストール作業がおこなわれる．以上の設定をもとにディスクを分割し，ファイルシステムを作成，配布物を展開する．
9. インストール終了後，ユーザはネットワークや特権ユーザのパスワード，ユーザの作成などを手動でおこなえる．
10. 再起動し，コンピュータからインストーラを取り外す．

3.1.5 基本システム

「基本システム」とは OS のソースツリーに格納されているプログラムの総称である．NetBSD のソースツリーは `ftp.netbsd.org/pub/NetBSD/NetBSD-7.1/source/sets/` で配布されている．またコンパイル済みの基本システムは `ftp.netbsd.org/pub/NetBSD/NetBSD-7.1/amd64/binary/` で配布されている．NetBSD のインストーラはディスクに基本システムを書き込んでいると解釈できる．

NetBSD のソースツリーは NetBSD 以外の Unix においてもビルドできるよう設計されている．図 5 は NetBSD のソースツリーをビルドしたさいの一部を図にしたものである．

NetBSD のソースツリーは Makefile でソースコード間の依存関係やコンパイル時のオプションを指定する．**usr/src** で `make build` と実行するとソースツリーがビルドされ生成物が **usr/obj** 下に作られる．しか

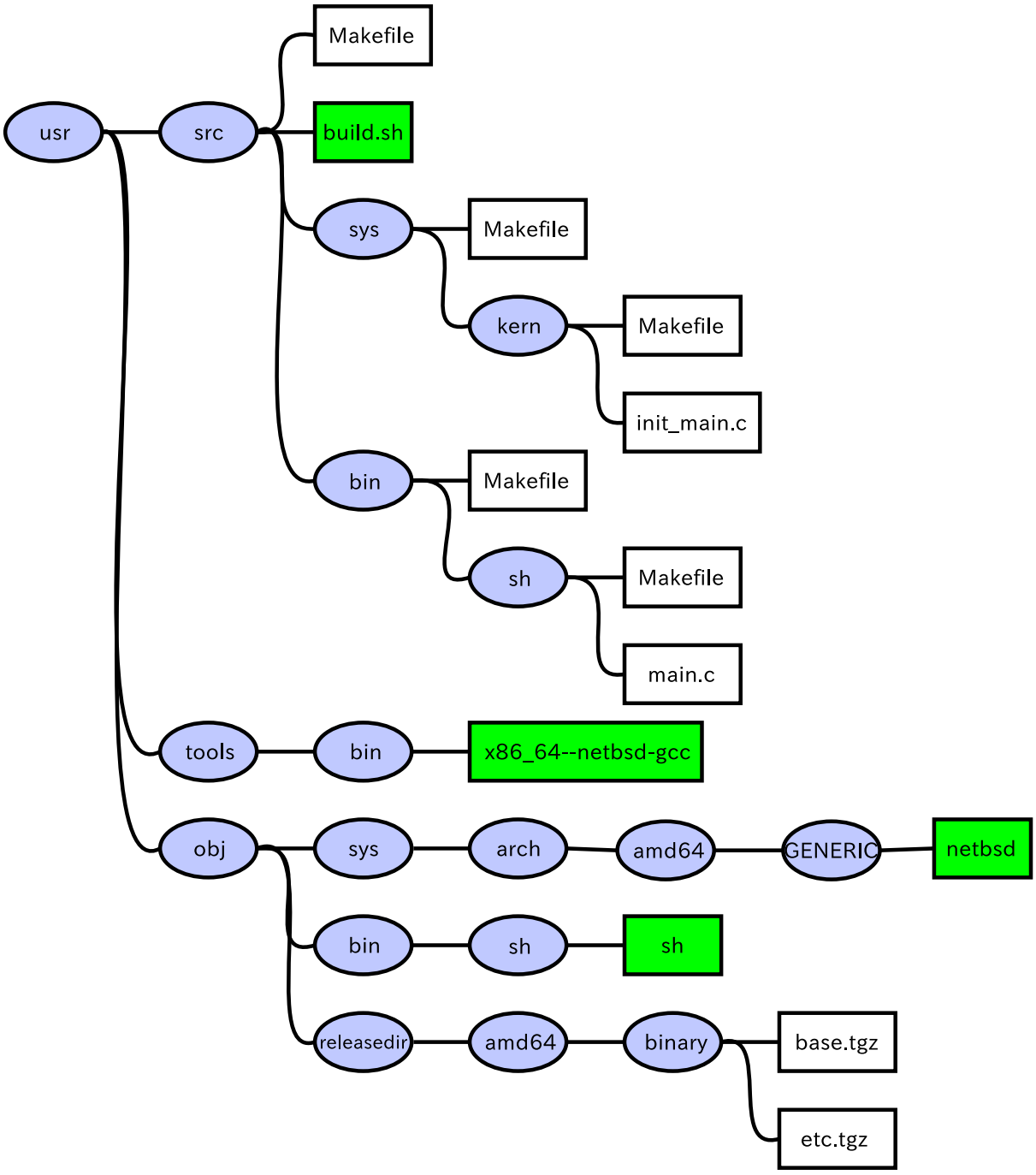


図 5　NetBSD ソースツリーの一部とビルド後の俯瞰図．はじめにソースツリーをビルドするためのプログラムが **usr/tools** へインストールされ，それを用いて基本システムが作られる．コンパイルされた基本システムは **usr/obj** へ置かれる．

```
nbasn1_compile      nblex               nbpwd_mkdb              x86_64--netbsd-cpp
nbawk               nblorder            nbrefer                 x86_64--netbsd-dbsym
nbcap_mkdb          nbm4                nbrpcgen                x86_64--netbsd-elfedit
nbcat               nbmake              nbsed                   x86_64--netbsd-fdisk
nbcksum             nbmake-amd64        nbslc                   x86_64--netbsd-g++
nbcompile_et        nbmakefs            nbsoelim                x86_64--netbsd-gcc
nbconfig            nbmakeinfo          nbstat                  x86_64--netbsd-gcc-4.8.5
nbcrunchgen         nbmakewhatis        nbstrfile               x86_64--netbsd-gcc-ar
nbctags             nbmandoc            nbsunlabel              x86_64--netbsd-gcc-nm
nbdb                nbmenuc             nbtbl                   x86_64--netbsd-gcc-ranlib
nbdisklabel         nbmkcsmapper        nbtexi2dvi              x86_64--netbsd-gcov
nbeqn               nbmkdep             nbtexi2pdf              x86_64--netbsd-install
nbfile              nbmkesdb            nbtexindex              x86_64--netbsd-ld
nbgenassym          nbmklocale          nbtic                   x86_64--netbsd-ld.bfd
nbgencat            nbmknod             nbtsort                 x86_64--netbsd-mdsetimage
nbgmake             nbmktemp            nbuudecode              x86_64--netbsd-nm
nbgroff             nbmsgc              nbyacc                  x86_64--netbsd-objcopy
nbhexdump           nbmtree             nbzic                   x86_64--netbsd-objdump
nbhost-mkdep        nbnroff             x86_64--netbsd-addr2line x86_64--netbsd-ranlib
nbindxbib           nbpax               x86_64--netbsd-ar       x86_64--netbsd-readelf
nbinstall-info      nbpaxctl            x86_64--netbsd-as       x86_64--netbsd-size
nbinstallboot       nbperf              x86_64--netbsd-c++      x86_64--netbsd-strings
nbjoin              nbpic               x86_64--netbsd-c++filt  x86_64--netbsd-strip
```

図 6　**usr/tools** の完全な一覧．このように一旦ビルド用のプログラムを準備することであらゆる環境で NetBSD をビルドできる．

し通常は **build.sh** と呼ばれる Bourne Shell で記述されたスクリプトを用いてソースツリーをビルドする．**build.sh** は OS やそのアーキテクチャに関わらず任意のアーキテクチャのバイナリを生成するためのスクリプトである [43].

はじめに **usr/src** にて `build.sh -O ../obj -T ../tools tools` を実行する．`tools` 命令はソースツリーをビルドするために必要なプログラム群 “tools” を生成する．`-O ../obj` はコンパイルして生成されたバイナリの格納先を指定している．`-T ../tools` は tools の格納先を指定している．`tools` 命令終了後，図 5 のように **usr/tools** 下へ **x86_64–netbsd-gcc** のようなバイナリが生成される．**usr/tools/bin** の完全な一覧は図 6 の通りである．ソースツリーのビルドを始める前にビルド用のプログラムを作っておくことで，OS やアーキテクチャを選ばずに NetBSD をビルドできる．

次に `build.sh -O ../obj -T ../tools release` を実行するとソースツリーがビルドされ，**usr/obj** 以下にコンパイルされたバイナリが作られる．また **usr/obj/releasedir** 以下には基本システムの配布物が作られる．

3.2　パッケージ

Unix では，複数のファイルやディレクトリを単一のファイルに格納する「アーカイブ」と呼ばれるファイル形式が存在する．**tar** (1) や **pax** (1) がアーカイブ作成プログラムとして知られている．

Unix 開発者はプログラム・ライブラリ・設定ファイル・マニュアルだけでなく，ソースコードをビルドした環境やインストール先のパスといったプログラムの情報が記述されたファイルも一緒にアーカイブに格納した

表 1　Unix システムとパッケージ形式およびパッケージ管理システムの一例.

名前	パッケージ形式	パッケージ管理システム
FreeBSD	txz	pkg
NetBSD	tgz	pkg_install
Debian	deb	apt
Ubuntu	deb	apt
Red Hat Enterprise Linux	rpm	yum
openSUSE	rpm	zypper

「パッケージ」と呼ばれるファイル形式を実装した．パッケージを操作するプログラムも実装することによって，システムへのインストール・アンインストールが容易になった．このプログラムは「パッケージ管理システム」と呼ばれる．

パッケージおよびパッケージ管理システムはシステムによって異なる．表 1 は Unix システムとパッケージ形式およびパッケージ管理システムの対応を一覧にしたものである．Ubuntu は Debian から派生した Linux ディストリビューションであり，パッケージ形式もパッケージ管理システムも同一のプログラムを採用している．一方，openSUSE は Red Hat Enterprise Linux が用いるパッケージ形式を採用しているが，パッケージ管理システムは異なるプログラムを採用しており，開発プロジェクトの方針によってシステムごとに違いがみえる．

3.3　BSD におけるパッケージ

BTL の Unix はカーネルとユーザランドをまとめてひとつのソースツリーにまとめ，テープに収録して配布していた．Berkeley の BSD もまた，BTL の Unix のソースツリーに変更を加える形で改良が重ねられ，テープで配布されていた．したがって現在の BSD ファミリは，単一のソースツリー上で基本システムをビルドするシステムが確立されている．BSD では基本システム以外の第三者製ソフトウェアのみをパッケージ形式で公式に管理している．

3.3.1　FreeBSD **ports** (7)

FreeBSD では **ports** (7) と呼ばれるパッケージ管理システムを用いて第三者製ソフトウェアのビルドとパッケージング・インストールをおこなう．**ports** (7) は図 7 のような木構造となっている．

ports (7) は通常**/usr/ports** へ配置される．ここで，ユーザが **git** (1) をシステムへインストールする場合，**/usr/ports/devel/git** へ移動し，`make install` を実行する．すると **ports** (7) は **git** (1) のソースコードをダウンロード・ビルドし，**git** (1) のパッケージを作成する．パッケージ **git-2.15.1.txz** は図 7 のように**/usr/ports/devel/git/work/pkg** へ生成される．最後に，**ports** (7) はこのパッケージを **pkg** (8) を用いてシステムへインストールする．

3.3.2　**pkg** (8)

FreeBSD のパッケージは **pkg** (8) を通じてシステムへのインストールやアンインストールがおこなわれる．また **pkg** (8) はネットワーク上のリポジトリを指定することで，手元にパッケージがない場合でもネッ

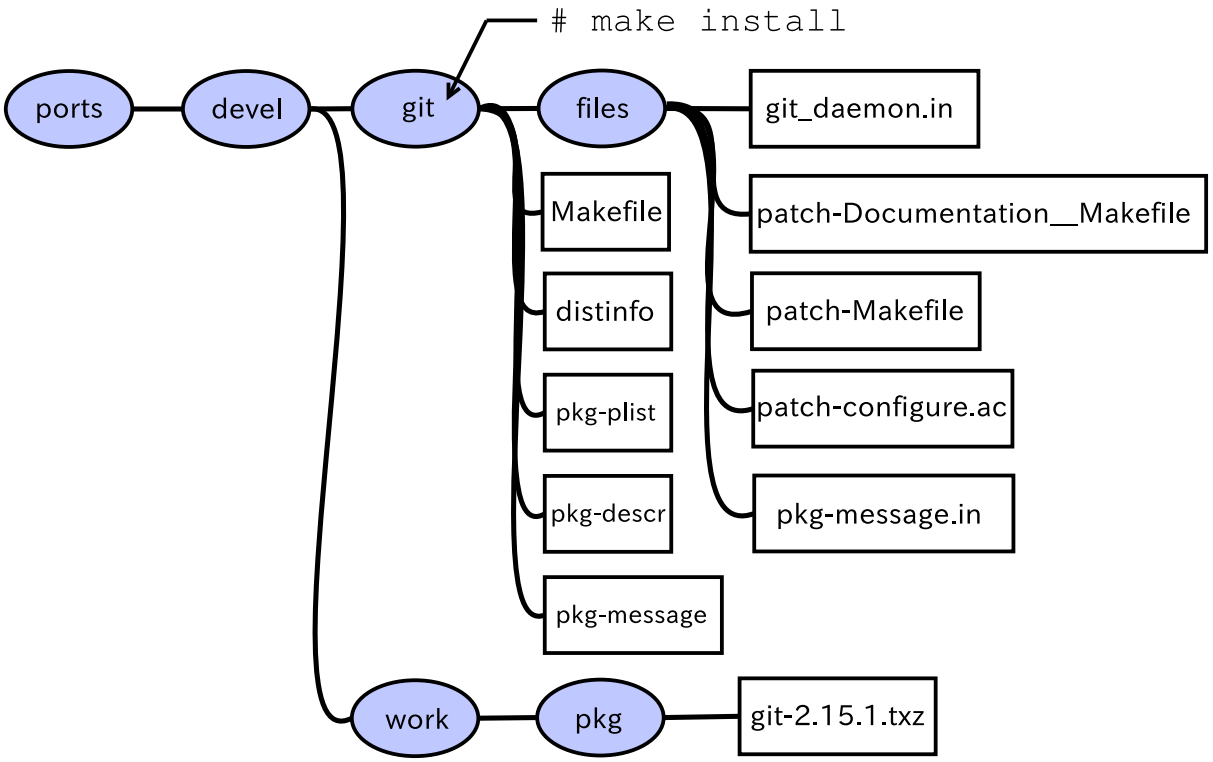


図 7　FreeBSD **ports** (7) の一部．**ports/devel/git** へ移動し `make install` を実行すると，**git** (1) が**/usr/local/bin** へインストールされる．

```
-rw-r--r--  0 root   wheel    2643  1 月  1  1970 +COMPACT_MANIFEST
-rw-r--r--  0 root   wheel   74079  1 月  1  1970 +MANIFEST
-rw-r--r--  0 root   wheel     227  1 月  4 10:35 /usr/local/share/licenses/git-2.15.1/catalog.mk
-rw-r--r--  0 root   wheel      81  1 月  4 10:35 /usr/local/share/licenses/git-2.15.1/LICENSE
-rw-r--r--  0 root   wheel   18092  1 月  4 10:35 /usr/local/share/licenses/git-2.15.1/GPLv2
-rwxr-xr-x  0 root   wheel 2109264  1 月  4 10:35 /usr/local/bin/git
-rwxr-xr-x  0 root   wheel  162750  1 月  4 10:35 /usr/local/bin/git-cvsserver
...
```

図 8　**ports** (7) の **git-2.15.1.txz** の内容．格納されているファイルの数が多いため，メタデータ以外は一部分を抜粋した．

トワークからパッケージをダウンロードしてインストール可能である．

pkg (8) が操作可能な，**ports** (7) によって作成されたパッケージ **git-2.15.1.txz** の内容は図 8 の通りである．

パッケージのメタデータは **+MANIFEST** であり，この省略版が **+COMPACT_MANIFEST** である．**+COMPACT_MANIFEST** は図 9 の通り JSON 形式でパッケージの情報が記載されている．

+MANIFEST は **COMPACT_MANIFEST** にファイルのチェックサムとインストール前後に実行されるシェルスクリプトが加わっている．

```
{"name":"git","origin":"devel/git","version":"2.15.1",
"comment":"Distributed source code management tool",
"maintainer":"garga@FreeBSD.org","www":"http://git-scm.com/",
"abi":"FreeBSD:11:amd64","arch":"freebsd:11:x86:64",
"prefix":"/usr/local","flatsize":27636630,"licenselogic":"single",
"licenses":["GPLv2"],
"desc":"Git is a free and open source distributed version control system
designed to\nhandle everything from small to very large projects with speed and
efficiency.\n\nWWW: http://git-scm.com/",
"deps":{"curl":{"origin":"ftp/curl","version":"7.57.0"},
"cvsps":{"origin":"devel/cvsps","version":"2.1_2"},
"expat":{"origin":"textproc/expat2","version":"2.2.1"},
"gettext-runtime":{"origin":"devel/gettext-runtime","version":"0.19.8.1_1"},
"p5-Authen-SASL":{"origin":"security/p5-Authen-SASL","version":"2.16_1"},
"p5-Error":{"origin":"lang/p5-Error","version":"0.17025"},
"pcre":{"origin":"devel/pcre","version":"8.40_1"},
"perl5":{"origin":"lang/perl5.24","version":"5.24.3"},
"python27":{"origin":"lang/python27","version":"2.7.14"}},
"categories":["devel"],"users":["git_daemon"],"groups":["git_daemon"],
"shlibs_required":["libpcre.so.1","libintl.so.8","libexpat.so.1","libcurl.so.4"],
"options":{"CONTRIB":"on","CURL":"on","CVS":"on","GITWEB":"on",
"GUI":"off","HTMLDOCS":"off","ICONV":"on","NLS":"on","P4":"on","PCRE":"on",
"PERL":"on","SEND_EMAIL":"on","SUBTREE":"on","SVN":"off"},
"messages":[{"message":"--------------------------------------------------------------------------\n
************************** GITWEB ************************************\n
If you installed the GITWEB option please follow these instructions:\n\nIn the directory
/usr/local/share/examples/git/gitweb you can find all files to\nmake gitweb work as a public
repository on the web.\n\nAll you have to do to make gitweb work is:\n1) Copy the files
/usr/local/share/examples/git/gitweb/* to a directory on\n
your web server (e.g. Apache2) in which you are
able to execute\n   CGI-scripts.\n2) In gitweb.cgi, adjust the variable $projectroot to point to\n
your git repository (that is where you have your *.git project\n   directories).\n
************************** GITWEB ************************************\n\n
************************** CONTRIB ***********************************\n
If you installed the CONTRIB option please note that the scripts are\ninstalled in
/usr/local/share/git-core/contrib. Some of them require\nother ports to be installed
(perl, python, etc), which you may need to\ninstall manually.\n
************************** CONTRIB ***********************************\n
--------------------------------------------------------------------------"}]}
```


図 9　**git-2.15.1.txz** の **+COMPACT_MANIFEST** の全文．本来は 1 行にまとまっているが，見やすさのために改行を加えた．

FreeBSD では，インストールされたパッケージの情報は**/var/db/pkg** にある SQLite データベースに蓄積され，**pkg** (8) を通じて参照される．

3.3.3　NetBSD **pkgsrc** (7)

NetBSD では第三者製ソフトウェアの管理に **pkgsrc** (7) を用いる．**pkgsrc** (7) は FreeBSD の **ports** (7) から派生したため構造は **ports** (7) と似ている．しかし **pkgsrc** (7) は **ports** (7) と異なり，NetBSD に限らず Linux ディストリビューションや MINIX といった他の Unix システム上でも動作する．

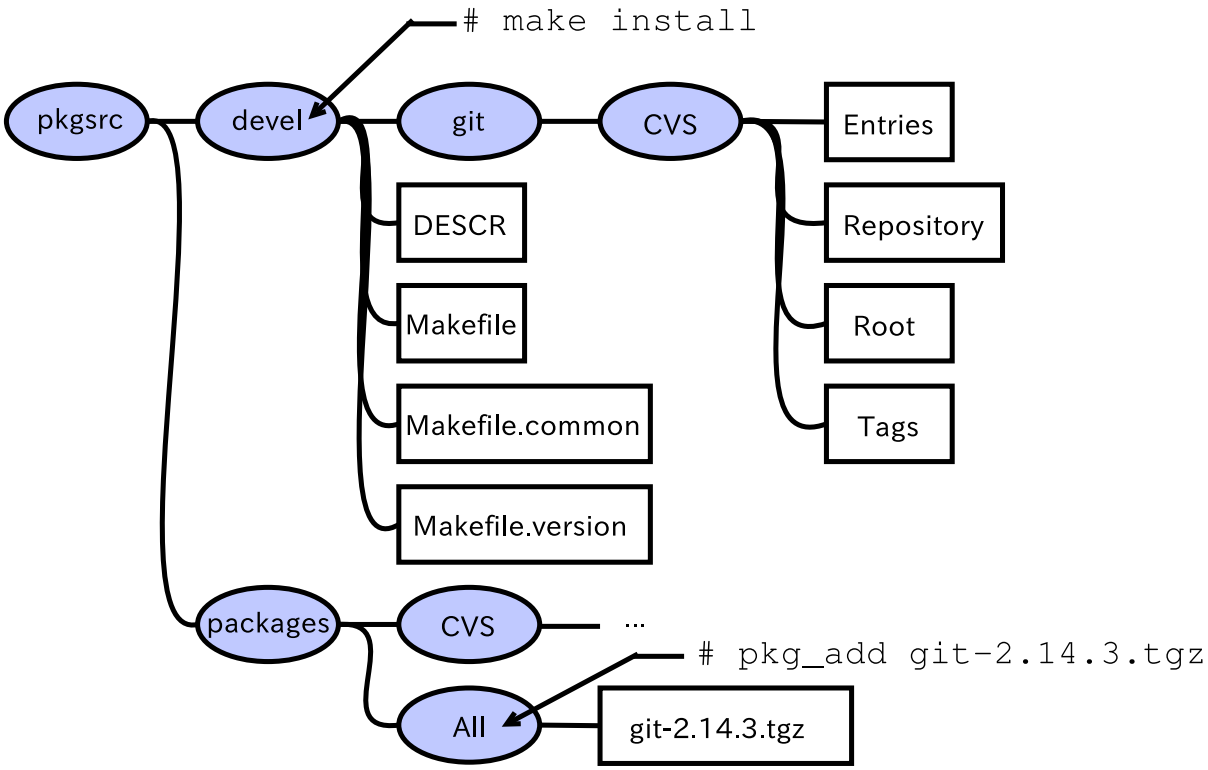


図 10　**pkgsrc** (7) の一部．**pkgsrc/devel/git** へ移動し `make install` を実行すると，**git** (1) のパッケージが **pkgsrc/packages/All** へ作られ，**pkg_add** (1) によって**/usr/pkg/bin** へインストールされる．

pkgsrc (7) は図 10 のような木構造となっている．

pkgsrc (7) は通常**/usr/pkgsrc** へ配置される．ここで，ユーザが **git** (1) をシステムへインストールする場合，**/usr/pkgsrc/devel/git** へ移動し，`make install` を実行する．**pkgsrc** (7) は **git** (1) のソースコードをダウンロード・ビルドし，**git** (1) のパッケージ **git-2.14.3.tgz** を作成する．**git-2.14.3.tgz** は **/usr/pkgsrc/packages/All** へ配置され，**pkgsrc** (7) は **pkg_add** (1) を使って **git-2.14.3.tgz** をシステムへインストールする．

CVS ディレクトリは Concurrent Versions System と呼ばれるバージョン管理プログラムが用いる情報が格納されている．**pkgsrc** (7) の動作とは直接関係がないためここでは説明を省略する．

3.3.4　pkg_install

pkgsrc (7) のパッケージを操作するには pkg_install と呼ばれるプログラム群を用いる．これは NetBSD の基本システムにインストールされている．また **pkgsrc** (7) では **pkgtools/pkg_install** を `make install` することで最新版の pkg_install をインストールできる．表 2 は各プログラムの説明の一覧である．

pkgsrc (7) の **git-2.14.3.tgz** の中身は図 11 の通りである．

ここで「+」から始まる +CONTENTS などのファイルは，pkg_install がパッケージの情報を読み取るためのメタデータである．このメタデータは，標準では**/var/db/pkg** へ格納される．

+CONTENTS はインストール先のプレフィックスとパッケージの依存関係およびインストールされるファ

表 2　pkg_install 群のプログラム名とその説明.

名前	説明
pkg_add (1)	システムへパッケージをインストールまたはシステム内のパッケージをアップグレードする.
pkg_admin (1)	パッケージの脆弱性情報やライセンスの確認をおこなう.
pkg_create (1)	パッケージを作成する.
pkg_delete (1)	システムへインストールされているパッケージを削除する.
pkg_info (1)	パッケージの情報を出力する.

```
$ tar vtf git-2.14.3.tgz
-rw-r--r-- root/wheel      418 2017-11-26 22:11 +CONTENTS
-r--r--r-- root/wheel       39 2017-11-26 22:11 +COMMENT
-r--r--r-- root/wheel      203 2017-11-26 22:11 +DESC
-rw-r--r-- root/wheel      266 2017-11-26 22:11 +BUILD_VERSION
-rw-r--r-- root/wheel      759 2017-11-26 22:11 +BUILD_INFO
-rw-r--r-- root/wheel        2 2017-11-26 22:11 +SIZE_PKG
-rw-r--r-- root/wheel       10 2017-11-26 22:11 +SIZE_ALL
$
```

図 11　**pkgsrc** (7) の **git-2.14.3.tgz** の中身.「+」から始まるファイルは pkg_install が読み取るパッケージのメタデータである.

イルの一覧が記載されている (図 12). オンラインマニュアルでは “packing list” と表されている [44].「@」から始まる行は **pkg_add** (1) のための命令を意味する. `@cwd /usr/pkg` とは, プログラムやマニュアルのインストール先が**/usr/pkg** 以下であると指定している. `@blddep git-base-2.14.3` は, このパッケージが git-base-2.14.3 を使ってビルドされたことを示している. `@pkgdep git-base>=2.14.3` は, このパッケージのインストールには git-base のバージョン 2.14.3 以上が必要であることを示している. **git-2.14.3.tgz** は git-base・git-contrib・git-docs・git-gitk にそれぞれ依存しているため, このパッケージをインストールすると自動的に依存されているこれらのパッケージがインストールされる.

+COMMENT と +DESC はパッケージの情報を記載している. +DESC は長い説明を, +COMMENT は短い説明を書く (図 13, 図 14).

+BUILD_VERSION はパッケージがビルドされるときに使われたファイルのバージョン情報が書かれている (図 15). バージョン情報はファイル名・バージョン・更新日時・作者名からなる.

+BUILD_INFO はパッケージを作成した環境やコンパイラに渡したオプションの情報が書かれている (図 16).

+SIZE_PKG と +SIZE_ALL はメタデータを除くパッケージのサイズと, インストールされるファイルの総容量を表している. 単位は Byte である. この情報があれば, システムのディスク容量に余裕がなかった場合インストール前にインストール作業を終了できる.

3.3.5　pkg-vulnerabilities

pkgsrc (7) では, パッケージの脆弱性情報の取得に **pkg_admin** (1) の `fetch-pkg-vulnerabilities` と

```
@cwd /usr/pkg
@name git-2.14.3
@blddep git-base-2.14.3
@pkgdep git-base>=2.14.3
@blddep git-contrib-2.14.3
@pkgdep git-contrib>=2.14.3
@blddep git-docs-2.14.3
@pkgdep git-docs>=2.14.3
@blddep git-gitk-2.14.3
@pkgdep git-gitk>=2.14.3
@blddep cwrappers-20170611
@comment git-2.14.3 has no files.
@cwd /usr/pkg
@ignore
+COMMENT
@ignore
+DESC
@ignore
+BUILD_VERSION
@ignore
+BUILD_INFO
@ignore
+SIZE_PKG
@ignore
+SIZE_ALL
```

図 12　**git-2.14.3.tgz** の +CONTENTS 全文.「@」から始まる行は **pkg_add** (1) が解釈する命令である.

```
GIT version control suite meta-package
```

図 13　**git-2.14.3.tgz** の +COMMENTS 全文.

```
This package is a meta package, collecting the components normally
expected to be installed for the GIT distributed version control
suite (the tool itself, the tk-based browser gitk, and the man pages).
```

図 14　**git-2.14.3.tgz** の +DESC 全文. +COMMENTS より長い.

```
devel/git/Makefile.common:      $NetBSD: Makefile.common,v 1.6 2017/08/16 20:21:06 wiz Exp $
devel/git/Makefile.version:     $NetBSD: Makefile.version,v 1.61.4.1 2017/11/25 08:49:05
bsiegert Exp $
devel/git/Makefile:     $NetBSD: Makefile,v 1.4 2015/07/10 17:32:57 khorben Exp $
```

図 15　**git-2.14.3.tgz** の +BUILD_VERSION 全文．紙面の都合上一部編集を加えた．

```
ABI=
BUILD_DATE=2017-11-26 13:11:26 +0000
BUILD_HOST=NetBSD amd64-nb7.netbsd.org 7.0 NetBSD 7.0 (LIBKVER)
#0: Tue Jan 19 00:00:00 UTC 2038 root@localhost:/sys/arch/amd64/compile/LIBKVER amd64
CATEGORIES=meta-pkgs devel
CFLAGS=-O2 -D_FORTIFY_SOURCE=2
CPPFLAGS=
FFLAGS=-O
LDFLAGS=   -Wl,-R/usr/pkg/lib
LICENSE=gnu-gpl-v2
LOCALBASE=/usr/pkg
MACHINE_ARCH=x86_64
MACHINE_GNU_ARCH=x86_64
MAINTAINER=pkgsrc-users@NetBSD.org
NO_BIN_ON_CDROM=
NO_BIN_ON_FTP=
NO_SRC_ON_CDROM=
NO_SRC_ON_FTP=
OBJECT_FMT=ELF
OPSYS=NetBSD
OS_VERSION=7.0
PKGGNUDIR=gnu/
PKGINFODIR=info
PKGMANDIR=man
PKGPATH=devel/git
PKGTOOLS_VERSION=20091115
PKG_SYSCONFBASEDIR=/usr/pkg/etc
PKG_SYSCONFDIR=/usr/pkg/etc
RESTRICTED=
SUPERSEDES=scmgit-[0-9]*
_PLIST_IGNORE_FILES=
_USE_DESTDIR=user-destdir
```

図 16　**git-2.14.3.tgz** の +BUILD_INFO 全文．

```
0
```

図 17　**git-2.14.3.tgz** の +SIZE_PKG 全文.

```
303738116
```

図 18　**git-2.14.3.tgz** の +SIZE_ALL 全文．依存しているパッケージの容量も含む.

audit および audit-pkg 命令を用いる．はじめに，fetch-pkg-vulnerabilities 命令で NetBSD Project 公式の FTP サーバから脆弱性情報のデータベースをダウンロードする.

```
# pkg_admin fetch-pkg-vulnerabilities
```

その後，audit 命令で脆弱性情報をすべて出力できる．システムにインストールされているパッケージの脆弱性情報のみを出力する場合は audit-pkg 命令を用いる.

```
# pkg_admin audit
Package python27-2.7.14 has a command-injection vulnerability,
see https://nvd.nist.gov/vuln/detail/CVE-2017-17522
Package xenkernel48-4.8.0nb1 has a denial-of-service vulnerability,
see https://xenbits.xen.org/xsa/advisory-214.html
Package xenkernel48-4.8.0nb1 has a denial-of-service vulnerability,
see https://xenbits.xen.org/xsa/advisory-213.html
Package xenkernel48-4.8.0nb1 has a sensitive-information-disclosure vulnerability,
see https://xenbits.xen.org/xsa/advi
sory-217.html
Package xenkernel48-4.8.0nb1 has a multiple-vulnerabilities vulnerability,
see https://xenbits.xen.org/xsa/advisory-218.html
...
```

パッケージの脆弱性情報のデータベースは**/var/db/pkg/pkg-vulnerabilities** に格納されている．**/var/db/pkg/pkg-vulnerabilities** の内容は図 19 の通り，

```
パッケージ名　　脆弱性の種類　　URL
```

という形式のデータベースが GnuPG という暗号プログラムで署名されている.

3.4　Linux におけるパッケージ

Debian や Red Hat Enterprise Linux といった Linux ディストリビューションは OS の構成要素すべてがパッケージ単位で管理され，BSD のように基本システムと第三者製ソフトウェアの明確な区別がない.

歴史的な経緯として，Linux はカーネルのみが開発され，**sh** (1) や **init** (1) といったユーザランド側のプログラムについては，GNU Project などが提供する第三者製ソフトウェアを集めなければならなかった．これは BSD の「基本システム＋第三者製ソフトウェア」という構成とは異なった OS の構成である.

本節では Debian のパッケージ形式に注目し，その内容について述べる.

```
-----BEGIN PGP SIGNED MESSAGE-----
Hash: SHA256

# $NetBSD: pkg-vulnerabilities,v 1.8360 2018/01/01 14:01:35 morr Exp $
#
#FORMAT 1.1.0
#
# Note: If this file format changes, please do not forget to update
# pkgsrc/mk/scripts/genreadme.awk which also parses this file.
#
# Note: NEVER remove entries from this file; this should document *all*
# known package vulnerabilities so it is entirely appropriate to have
# multiple entries in this file for a single package, and to contain
# entries for packages which have been removed from pkgsrc.
#
# New entries should be added at the end of this file.
#
# Please ask pkgsrc-security to update the copy on ftp.NetBSD.org after
# making changes to this file.
#
# The command to run for this update is "make upload", but it needs
# access to the private GPG key for pkgsrc-security.
#
# If you have comments/additions/corrections, please contact
# pkgsrc-security@NetBSD.org.
# package               type of exploit         URL
pine<4.30               remote-user-shell       http://www.securityfocus.com/bid/1709
pine<4.21nb1            denial-of-service       http://www.securityfocus.com/advisories/2646
imap-uw<4.7c6           denial-of-service       http://www.securityfocus.com/advisories/2646
screen<3.9.5nb1         local-root-shell        http://www.securityfocus.com/advisories/2634
ntop<1.1                remote-root-shell       http://www.securityfocus.com/advisories/2520
...
#CHECKSUM SHA1 abe70765a8141889be8e93aadd13fda8227ce03b
#CHECKSUM SHA512
a6d0009afa914551cc77dc3e4186895788c595d8a2e8e17ee5c919c85deb7cae14ce5e911d3ae1eb
a6fad75b37572da5b30eaecde07f204dff1b5fca91ab655d
-----BEGIN PGP SIGNATURE-----
Version: GnuPG v2

iQIcBAEBCAAGBQJaSk9oAAoJEHa4lU5BBt7/uhYP/0FiQbXsRj8yMMNSZjnhhZTI
hBKzHm8BdV9t8Po6GVyeO+Qi1q1nDm96fv+oZtqJnrMzD8INX/0c5XBPdU3Dxemh
VbUbu8f6RqZrN1Cutr4maqWyNUkIQUKybWrFStzVSUJXWJNOHZJB4tJnB5dL0F35
UBOdxeIV5UWOWlBjE3jXOu9Vkd9dAGXmjZmZ9tBar7f9aHUxPW2wfyaNA7+hkbWZ
DkT7oNFgzRvZPZ5T5bwIloS5pSm+8iF7qrfahYiB2X17DyiFD35BgPFqLK9tQ4Gz
ZeOctLUQ1+PENYHoxhbJWNqtdbahv5DbMpuOPTaJ7pHCRKZEjtpJKs1TBOdhCG5Q
VeW9lyFmz8Ao+rZH+hmZ1jz2f+csEhX+ZtzHPvEe9azKIGOeia7NKMKNiBWLgwmr
LF1J+hJnGSzl8jL1yLjigB1e8zRefVmb/v4cfon1woMOADs7nwglFWVUqNroEK7k
k44HklxnfPjWErSPIIIcdRd4OzlhOupyiVI59mtDh/yG6REGvWW2ml6MK97b71ho
BWGnzYi6vR/Gxue0JcSZvtXYmr5Sl74w4cFN75kjprWWW66rcN1KgbEFXQTyK5vE
upwvFVEwrqKxWyZnecYbi9iJBQwois6GI+nWZUzonkR5uCK79ZBQdLKA9SREo+XH
uLXAfLyavG9xu3EomTYy
=Adf5
-----END PGP SIGNATURE-----
```

図 19　**/var/db/pkg/pkg-vulnerabilities** の内容を **zcat** (1) を用いて表示させたもの．GnuPG で署名されており，パッケージ名と脆弱性の種類および URL の対応が並ぶ．

```
$ ar t git_2.1.4-2.1+deb8u5_amd64.deb
control.tar.gz
data.tar.xz
debian-binary
$ tar zvtf control.tar.gz
drwxr-xr-x root/root         0 2017-09-26 04:32 ./
-rw-r--r-- root/root     32922 2017-09-26 04:32 ./md5sums
-rwxr-xr-x root/root      3143 2017-09-26 04:32 ./preinst
-rw-r--r-- root/root        34 2017-09-26 04:32 ./conffiles
-rw-r--r-- root/root      1702 2017-09-26 04:32 ./control
-rwxr-xr-x root/root       478 2017-09-26 04:32 ./prerm
-rwxr-xr-x root/root      1570 2017-09-26 04:32 ./postinst
-rwxr-xr-x root/root      1080 2017-09-26 04:32 ./postrm
$ tar Jvtf data.tar.xz
drwxr-xr-x root/root         0 2017-09-26 04:32 ./
drwxr-xr-x root/root         0 2017-09-26 04:32 ./etc/
drwxr-xr-x root/root         0 2017-09-26 04:32 ./etc/bash_completion.d/
-rw-r--r-- root/root       439 2017-09-26 04:32 ./etc/bash_completion.d/git-prompt
drwxr-xr-x root/root         0 2017-09-26 04:32 ./var/
drwxr-xr-x root/root         0 2017-09-26 04:32 ./var/lib/
drwxr-xr-x root/root         0 2017-09-26 04:32 ./var/lib/git/
drwxr-xr-x root/root         0 2017-09-26 04:32 ./usr/
drwxr-xr-x root/root         0 2017-09-26 04:32 ./usr/bin/
-rwxr-xr-x root/root   1680768 2017-09-26 04:32 ./usr/bin/git
...
$
```

図 20　**git_2.1.42.1+deb8u5_amd64.deb** の中身を **ar** (1) と **tar** (1) で一覧表示したもの．**data.tar.xz** の一覧は省略した．

3.4.1　Debian

Debian は Debian binary package format(deb) と呼ばれる形式のパッケージを扱い，パッケージ管理システム **dpkg** (1) を通じてパッケージを操作する．**dpkg** (1) 以外の高度な deb パッケージ管理システムとして，**apt** (8) や **apt-get** (8) がある．

deb 形式の **git_2.1.42.1+deb8u5_amd64.deb** の中身は図 20 の通りである．**dpkg** (1) が読み取るメタデータは **control.tar.gz** に格納されており，プログラムや設定ファイル・マニュアルは **data.tar.xz** に格納されている．**debian-binary** は deb 形式のバージョン番号を表している [45].

md5sums は **data.tar.xz** に格納されているファイルのチェックサムの一覧である (図 21)．この値によって，ファイルの同一性を確認できる．

preinst と **prerm** はパッケージのインストール前に実行されるシェルスクリプトが書かれている．同じく，**postinst** と **postrm** はパッケージのインストール後に実行されるシェルスクリプトが書かれている．これらのシェルスクリプトは特定の引数が与えられて実行されるため，引数の内容に応じて複数の処理を記述で

```
7baac5c3ced94ebf2c0e1dde65c3b1a6  etc/bash_completion.d/git-prompt
d485b1a6aa820c4d36f8a65cca42b29e  usr/bin/git
29d0daf980b334f822679a29b3466ed3  usr/bin/git-upload-pack
3aef9cc2775a91b5bc00335783fde031  usr/bin/git-shell
a08351f609679fb70f10546c12494919  usr/share/bash-completion/completions/git
...
```

図 21　**md5sums** の中身．5 行目以降は省略した．

```
/etc/bash_complention.d/git-prompt
```

図 22　**conffiles** の中身．設定ファイル名が絶対パスで書かれている．

きる [46][47][48][49].

conffiles にはそのパッケージに格納されているプログラムの設定ファイルの絶対パスが記載されている (図 22)[50].

control には図 23 のようにパッケージ名やバージョン・対応アーキテクチャ・メンテナ情報・プログラムのサイズ・依存関係・開発プロジェクトのホームページ・パッケージの説明が書かれている．**control** の情報は `apt show` を実行することで確認できる．

3.5　ライセンス

FLOSS には著作権表示や利用の条件などを明示するなんらかのライセンスが適用されている．本節では FLOSS ライセンスとして代表的な GNU GPL と BSD License について説明する．

3.5.1　GNU GPL

GNU GPL は FSF のフリーソフトウェア運動のために考案されたライセンスである．GNU GPL が適用されている代表的なプログラムには Linux カーネルがある．GNU GPL をプログラムに適用することで，自分もしくは第三者によってリリースされるそのプログラムの改良版に GNU GPL を適用させ，Free Software であるようにさせることが可能である [51]．GNU GPL の第 1 版は 1989 年に発行された．改訂版の第 2 版は 1991 年に発行された．最新版は 2007 年に発行された第 3 版である．

GNU GPL は派生物にまで GNU GPL を適用させ，ソースコードの公開を義務付けさせるため企業によるライセンス違反とそれに対する訴訟が起きた事例も存在する．2009 年 12 月には製品に使われている “BusyBox” と呼ばれる組み込み向けプログラムの利用がライセンス違反であるとして Samsung Electronics や Best Buy ら 14 社が訴訟され，2010 年 7 月に製品の販売禁止と 9 万ドルの損害賠償および 4 万 7 千ドルの訴訟費用の支払いが命じられた [52]．また 2013 年には FANTEC のメディアプレーヤに利用されているファームウェアのソースコードの公開に関する訴訟が Harald Welte によって起こされ，FANTEC が敗訴した [53].

```
Package: git
Version: 1:2.1.4-2.1+deb8u5
Architecture: amd64
Maintainer: Gerrit Pape <pape@smarden.org>
Installed-Size: 21946
Depends: libc6 (>= 2.16), libcurl3-gnutls (>= 7.16.2), libexpat1 (>= 2.0.1),
libpcre3 (>= 1:8.35), zlib1g (>= 1:1.2.0), perl-modules, liberror-perl,
git-man (>> 1:2.1.4), git-man (<< 1:2.1.4-.)
Recommends: patch, less, rsync, ssh-client
Suggests: gettext-base, git-daemon-run | git-daemon-sysvinit, git-doc, git-el,
git-email, git-gui, gitk, gitweb, git-arch, git-cvs, git-mediawiki, git-svn
Breaks: bash-completion (<< 1:1.90-1), cogito (<= 0.18.2+), git-buildpackage (<< 0.6.5),
git-core (<< 1:1.7.0.4-1.), gitosis (<< 0.2+20090917-7), gitpkg (<< 0.15),
gitweb (<< 1:1.7.4~rc1), guilt (<< 0.33), stgit (<< 0.15), stgit-contrib (<< 0.15)
Replaces: git-core (<< 1:1.7.0.4-1.), gitweb (<< 1:1.7.4~rc1)
Provides: git-completion, git-core
Section: vcs
Priority: optional
Multi-Arch: foreign
Homepage: http://git-scm.com/
Description: fast, scalable, distributed revision control system
 Git is popular version control system designed to handle very large
 projects with speed and efficiency; it is used for many high profile
 open source projects, most notably the Linux kernel.
 .
 Git falls in the category of distributed source code management tools.
 Every Git working directory is a full-fledged repository with full
 revision tracking capabilities, not dependent on network access or a
 central server.
 .
 This package provides the git main components with minimal dependencies.
 Additional functionality, e.g. a graphical user interface and revision
 tree visualizer, tools for interoperating with other VCS's, or a web
 interface, is provided as separate git* packages.
```

図 23　**control** 全文．パッケージの情報がそれぞれ記載されている．

3.5.2　BSD License

BSD License は UCB によって策定されたライセンスである．1993 年に 4 条項の BSD License が発行された．このライセンスは「無保証・無責任であること」また「以下の条項に承諾すること」を条件に再頒布または使用が許可される [54]．

1. ソースコードの再頒布には BSD License の著作権表示と免責条項を記載する．
2. バイナリ形式での頒布には BSD License の著作権表示と免責条項を文書に記載する．

3. BSD License のソースコードを用いた製品の広告には，UCB およびそれに貢献した人が開発したソフトウェアが含まれていることを謝辞として記載する．
4. 許可なく大学名または貢献した人の名前を広告に利用することはできない．

1998 年，3 番目の条項を取り除いた 3 条項 BSD License が発行され，1999 年，最初の 2 つのみを条項として用いる 2 条項 BSD License が発行された [55]．

BSD License は GNU GPL のように派生物のソースコードを公開させる義務を課さないため，改変や再頒布をおこなうさいはソースコードを公開せずバイナリ形式のまま頒布できる．

4　BSD のパッケージ化の背景と目標

4.1　パッケージの利点

OS の構成要素が Debian のようにすべてパッケージ単位で分割・管理されていることは，BSD のような単一のソースツリーによる基本システムの管理に比べて利点が存在する．

はじめに，OS の構成要素がすべてパッケージであれば，システムの構成要素を細かく管理できる．たとえば，Debian は essential なパッケージとそうでないパッケージを区分している．essential はシステムに必ずインストールされていなければならないことを意味する．essential パッケージの一覧は図 24 の通りである．これらすべてのパッケージをインストールすると，合計 5.3MB のディスク容量が求められる．

一方，NetBSD は配布物のうち **base.tgz** と **etc.tgz** がシステムに必須である．これらふたつをインストールすると，合計 186MB のディスク容量が求められる．

このように，パッケージで管理されているシステムはそうでないシステムに比べ必要最小限の容量が大きく異なる．

また OS の構成要素がすべてパッケージであれば，システムのアップデートがパッケージでないシステムと比べ高速である．

Debian では `apt update` でパッケージのバージョン情報を更新し，`apt upgrade` でシステム内の更新可能なパッケージをアップデートできる．また `apt dist-upgrade` を用いればバージョンアップデートも可能である．

BSD においても第三者製ソフトウェアであればパッケージで管理されているが，基本システムはソースツリーをビルドするかコンパイル済みのバイナリを収めたアーカイブをダウンロードし，展開しなければならない．これは FreeBSD であれば **freebsd-update** (8)，NetBSD であれば **build.sh** または **sysinst** (8) を用いる作業に相当する．

このようなシステムアップデート方法の違いは，特にシステムの一部分のみを更新するさいに重要となる．基本システムのプログラムやライブラリに脆弱性が見つかり，早急にアップデートをおこなわなければならない場面を想定する．

Debian であれば該当するパッケージを **apt** (8) を用いてアップデートするだけである．しかし NetBSD であれば NetBSD Project 公式の FTP サーバ, たとえば `ftp.netbsd.org/pub/NetBSD/NetBSD-7.1/amd64/binary/sets/` から最新のアーカイブをダウンロードするか，最新のソースツリーを **build.sh** を通じてビルドしなければならない．FreeBSD も同様に，**freebsd-update** (8) を用いてバイナリをダウンロードするか，ソースツリーをビルドすることで基本システムのアップデートが可能となる．しかしこのような BSD のアッ

```
base-files  base-passwd bash               bsdutils       coreutils dash
debianutils diffutils   dpkg               e2fsprogs      findutils grep
gzip        hostname    init-system-helpers libc-bin       login     ncurses-base
ncurses-bin perl-base   sed                sysvinit-utils tar       util-linux
```

図 24　Debian の essential パッケージの一覧．

プデート手法では，システムのどの箇所に変更が加わり，変更がない箇所がどこであるかを把握することが難しい．パッケージ単位のアップデートであれば，どのパッケージをいつアップデートしたかの記録を残すことが可能である．

パッケージ管理にはこのような技術的な利点だけでなく，パッケージひとつひとつの動作を保証し，アップデートを提供し，ユーザサポートをおこなうビジネスモデルを確立できる商業的な利点も存在する．Red Hat は Red Hat Enterprise Linux のバイナリとソースコードにアクセスする権利やソフトウェアのアップデートをおこなう権利をサブスクリプションとして提供し，利益を得ている．

以上，システムの構成要素をすべてパッケージで管理することの利点を述べた．しかしパッケージの利点は BSD の開発プロジェクトも把握しており，基本システムをパッケージに分割する機能の開発がおこなわれている．

4.2 FreeBSD PkgBase

FreeBSD では 2016 年 4 月から “PkgBase” と呼ばれる機能が公式に追加された [56]．PkgBase は基本システムのすべての構成要素を **pkg** (8) から操作可能な 782 個のパッケージに分割する．

PkgBase は未だベータ版であり，FreeBSD Project はリポジトリを提供していない．すなわち Debian のように **apt** (8) からすべてをアップデート可能な運用体制ではなく，ユーザ自身が FreeBSD のソースツリーをビルドして手動でパッケージを作成する必要がある．

ソースツリー上で `make buildworld` と `make buildkernel` および `make packages` を順番に実行すると，**/usr/obj/usr/src/repo/**以下にリポジトリが作成される．筆者の環境では**/usr/obj/usr/src/repo/FreeBSD:12:amd64/latest/**にパッケージが作成された．このパッケージの形式は **ports** (7) によるパッケージの形式と変わらず，メタデータに **+MANIFEST** と **+COMPACT_MANIFEST** が使われる．

4.3 NetBSD syspkg

NetBSD では 2002 年に “syspkg” と呼ばれる機能が公式に追加されている [57]．syspkg は基本システムのすべての構成要素を **pkg_add** (1) や **pkg_delete** (1) などで利用可能なパッケージの形で 681 個に分割する．

syspkg によるパッケージは NetBSD Project 公式の FTP サーバにて配布されていない．PkgBase と同様に Debian のような運用体制になく，ユーザ自身で **build.sh** を使いパッケージを作成する必要がある．

syspkg は NetBSD のソースツリー下で `build.sh release` を実行したのちに `build.sh syspkgs` を実行することで, amd64 アーキテクチャ向けパッケージは標準では**/usr/obj/releasedir/amd64/binary/syspkgs** へ生成される．しかし syspkg によるパッケージの形式は，**pkgsrc** (7) によるパッケージの形式と異なる箇所が存在する．図 25 は syspkg によって作成された **base-sys-root-7.1.0.20171201.tgz** の内容である．

ここで **+PRESERVE** とは，そのパッケージが削除されないことを意味するメタデータである [44]．内容は図 26 のようにパッケージ名が記載されている．一方で，**pkgsrc** (7) のパッケージには存在した **+BUILD_VERSION** · **+INSTALL** · **+DEINSTALL** · **+SIZE_PKG** · **+SIZE_ALL** といったメタデータが syspkg のものには存在しない．

```
-r--r--r--  1 root     wheel       4434 Dec  4 19:20 +CONTENTS
-r--r--r--  1 root     wheel         36 Dec  4 19:20 +COMMENT
-r--r--r--  1 root     wheel         70 Dec  4 19:20 +DESC
-r--r--r--  1 root     wheel         81 Dec  4 19:20 +BUILD_INFO
-r--r--r--  1 root     wheel         29 Dec  4 19:20 +PRESERVE
drwxr-xr-x  2 root     wheel          0 Dec 15 20:14 .
drwxr-xr-x  2 root     wheel          0 Dec  1 20:39 altroot
drwxr-xr-x  2 root     wheel          0 Dec  2 02:40 bin
drwxr-xr-x  2 root     wheel          0 Dec  2 02:50 dev
drwxr-xr-x  2 root     wheel          0 Dec  1 20:39 dev/altq
drwxr-xr-x  2 root     wheel          0 Dec  1 20:39 dev/fd
```

図 25　syspkg による **base-sys-root-7.1.0.20171201.tgz** の内容.

```
base-sys-root-7.1.0.20171201
```

図 26　**base-sys-root-7.1.0.20171201.tgz** の **+PRESERVE** の内容.

4.4　本研究の目標

第 1.1 節と第 1.2 節で述べたように，Unix は世界中のサーバや製品に用いられる．一方で Unix のプログラムの脆弱性を対象としたクラッキングも報告されている．脆弱性が発見された場合は対象のプログラムやライブラリを迅速にアップデートしなければ，サービスの不具合やクラッキングの温床となる可能性も考えられる．

アップデートのさい，BSD Unix のようなソースツリーのビルドや配布物のダウンロードと展開よりも，Debian のようなパッケージ管理システムを通じたパッケージによるアップデート手法のほうが高速かつ単純である．

Debian や RHEL の，システム全体をパッケージ単位で管理する方式は，ユーザの利便性とアップデートの効率性の点からみれば BSD のシステム構成よりも優れている．しかし，Linux ディストーションに用いられているカーネルまたは GNU Project によるユーザプログラムのライセンスは，伝播性のある GNU GPL である．第 3.5.1 節で述べたように，GNU GPL のソースコードを用いた製品には訴訟の危険性がある．

OS の機能拡張および商用利用を考慮すると，制約が緩い BSD License で提供される BSD Unix の利用が訴訟リスクを避ける上で望ましい．BSD Unix が商用利用されることを考えれば，システム管理やシステム開発の効率化を狙う上で，基本システムのパッケージ化は急務である．

FreeBSD は PkgBase を開発中であり，NetBSD には syspkg が存在するが，2018 年 1 月の時点では両者ともにパッケージを公式に配布するに至っていない．システムの細分化とインストールまたはアップデートの高速化を目的としたパッケージをネットワーク上のサーバからダウンロードできなければ，パッケージの作成がユーザに委ねられてしまうため，基本システムをパッケージ管理する利点が失われてしまう．

特に syspkg は “There has been a lot of work in this area already, but it has not yet been finalized” [57]

とあるように，2002 年に開発が始められたのにも関わらず未だ完成していない．またユーザによる問題報告データベースにおいても，2012 年に提出された syspkg の修正パッチが 2018 年現在でも放置されたままであり [58]，開発が停滞している状態にある．

NetBSD は 58 種類の CPU アーキテクチャをサポートしている OS である．ベンダがどのような用途でどの CPU アーキテクチャを選択するかは不明であるため，サポートしている CPU アーキテクチャ数が多いシステムは商用利用にとって有効である．その中には amd64 や i386 に比べて処理が遅いアーキテクチャや，ハードウェア資源に限りがあるアーキテクチャも含まれている．arm や m68k・sh3・powerpc が例として挙げられる [59]．

セキュリティアップデートのさい，これらのような遅いアーキテクチャ上で **build.sh** を実行することは現実的な対応とは言い難い．**build.sh** はビルド環境とは異なる CPU アーキテクチャ用のバイナリをビルドできるが，既存のアップデート手法に比べればパッケージによるアップデートの支援は NetBSD がサポートする多くのアーキテクチャにとって利点が大きいと考えられる．

そこで我々は syspkg の代替となるシステムを新規に開発し，syspkg の代わりに NetBSD の基本システムをパッケージ管理することを目指した．我々が開発したシステムは “basepkg” と呼び，syspkg のように NetBSD の基本システムをパッケージ化する．basepkg は syspkg には無かった機能を実装し，basepkg を用いたパッケージの配布サーバも試験的に運用している．一方で basepkg の開発およびパッケージ配布サーバの運用を通じて，解決すべき問題が多く残されていることもわかった．

5 basepkg

本章では syspkg の代替として開発した NetBSD の基本システムをパッケージに分割するフレームワークである basepkg[60][**?**] について述べる．basepkg は 2 条項 BSD License[61] を適用した OSS であり，2016 年 10 月 26 日から `github.com/user340/basepkg` にて公開し開発をおこなっている．また 2017 年 5 月 19 日には **pkgsrc** (7) の Work-in-Progress 版である **pkgsrc-wip**[62] に登録し，パッケージとして配布している．

5.1 新規開発に至った理由

NetBSD の基本システムのパッケージ化をおこなうに際し，我々は syspkg を修正するのではなく，以下の理由から新規に basepkg を開発することに決めた．

第一に，NetBSD の開発プロセスの複雑さが挙げられる．syspkg は NetBSD の基本システムに組み込まれた機能であるため，syspkg の修正は NetBSD のソースツリーに手を加える必要がある．しかし NetBSD は，コミット権を持つ少数の開発者 (コミッタ) のみがソースツリーへ変更を加えることができる．コミッタになるには，はじめにコミッタによる推薦を受け，開発者申請をおこない，最大 14 ヶ月間の審査に通らなければならない [63]．

コミッタでない開発者は，ソースコードの修正などの要求を **send-pr** (1) を通じておこなえる．このプログラムを通じてパッチや新規ファイルをコミットするようコミッタに申請できるが，この要求を通すか通さないかはコミッタ次第であるため確実ではない．現に，2012 年に送られた syspkg に関する修正パッチは未だコミットされていない．

このような伽藍方式の開発プロセスは開発以外の政治的な活動に時間を取られてしまい，確実に修正パッチの反映がされる保証もない．したがって我々は syspkg を直接修正するのではなく，新規開発を決定した．

第二に, syspkg 自体のドキュメントの少なさと構造の複雑さが挙げられる．syspkg の構造やフローチャートといった実装に関する情報を残したドキュメントは存在しない．syspkg の概要やプロジェクトの目標が書かれたテキストファイルは **usr/src/distrib/syspkg/notes** に存在するが，どのようなフローでどのような処理がおこなわれるのかはソースコードを読まなければ不明である．しかし syspkg は **usr/src/distrib/sets** と **usr/src/distrib/syspkg** に存在する複数の Makefile とシェルスクリプトから構成されており，syspkg の処理の流れを追うことは困難であった．

そこで，筆者は syspkg と同等の動作をするスクリプトを開発し直すことで，syspkg がどのような流れでパッケージを作成するのかという大まかな全体像を掴む作業と，NetBSD のパッケージの形式について把握する作業を並行しておこなうことにした．既存のシステムを新しく作り直す行為は「車輪の再発明」とも揶揄され批判の対象となり得るが，Kernighan と Plauger が「だめなプログラムを修正するのはやめて、全部書きなおそう」[64, pp. 102] と述べたように，場合によっては、はじめから書き直す手法が効果的であるという指摘も存在する．

5.2 開発言語および POSIX 中心主義

syspkg は Makefile とシェルスクリプトで開発されている．basepkg は Makefile を使わずシェルスクリプトのみで開発した．

NetBSD のソースツリーは **build.sh** を用いてあらゆる Unix 環境でビルド可能である．syspkg もあらゆ

```
CVS             comments        listpkgs        makesums        sets.subr
Makefile        culldeps        lists           maketars        sort-list
README          deps            makeflist       metalog.subr    syspkgdeps
TODO            descrs          makeobsolete    mkvars.mk       versions
attrs           getdirs.awk     makeplist       regpkg
checkflist      join.awk        makesrctars     regpkgset
```

図 27　NetBSD の **usr/src/distrib/sets** の内容．syspkg ではこれらのスクリプトが使われパッケージが作成される．

る Unix 環境で動作し，NetBSD のパッケージを作成できる．したがって syspkg と同等の機能を実現するには，どのような Unix 環境でも動作する実装をおこなわなければならない．

ここで 2 通りの方法が考えられる．第一に，ソースコードを C++・Java・Python のような高級なプログラミング言語で書き，コンパイラやインタプリタのソースコードと共に配布する方法がある．これは **build.sh** のように，はじめにビルド用のプログラムを作らせてから NetBSD のソースツリーをビルドするブートストラップ式の方法である．第二に，Unix の国際規格である Portable Operating System Interface (POSIX) にて定められているプログラムを中心に使うシェルスクリプトを書く方法がある．POSIX 規格を中心としたプログラミング手法は「POSIX 中心主義」と言われ，システムの持続性や互換性といった効果検証がおこなわれている [65].

我々は POSIX 規格中心のシェルプログラミングによる実装を選んだ．高級言語による実装，特に C++ や Java・Golang などコンパイルをおこないバイナリへ変換するプログラミング言語は実行速度が速いと言われている．しかしシェルスクリプトでも **awk** (1) や **find** (1)・**xargs** (1) で扱える内部ループを中心とした実装をおこなえば実用的な実行速度が出せるという報告が松浦らによりなされている [65]．またシェルスクリプトはコンパイルをおこなう必要がなく，プログラムのテストをコンパイルが必要な言語より簡単におこなうことができる．

また syspkg のようにファイルを分散させず，単一のファイルに処理をすべて記述した．Unix 哲学の観点からすれば，「スモール・イズ・ビューティフル」[14, pp. 14] であり，「一つのプログラムには一つのことをうまくやらせる」[14, pp. 23] ことが Unix らしい開発手法であるといえる．また Kernighan らも『ソフトウェア作法』において，小さいプログラムを複数作り，それらをつなぎ合わせることでテキストエディタを開発するプロセスを読者に示した [66]．ファイルを複数個に分散させてそれらをつなぎ合わせる syspkg の設計は，Unix 哲学的には正しく，また実践されてきた手法であるといえる．

しかしこのような設計はプログラムの実行の流れを上から下へと追っていくことが困難になる．図 27 は **usr/src/distrib/sets** の内容であるが，これらのスクリプトが Makefile の中で順に呼ばれていくことになる．

小さい部品を組み上げてプログラムを書くことは重要であるが，プログラムの処理の流れを上から下へ逐次的に読めることもまた重要であると Kernighan や Boswell らによって指摘されている [64][67]．今回の basepkg の開発では，Unix 的な設計思想よりもプログラムの可読性に重点を置き，単一のファイルにスクリプトをすべてまとめることとなった．

```
# cd /usr
# ftp ftp://ftp.netbsd.org/pub/pkgsrc/stable/pkgsrc.tar.gz
...
# tar zxf pkgsrc.tar.gz && rm -f pkgsrc.tar.gz
...
# cd pkgsrc
# git clone git://wip.pkgsrc.org/pkgsrc-wip.git wip
...
# cd wip/basepkg
# make install
...
#
```

図 28　**pkgsrc** (7) と **pkgsrc-wip** をインストールし，basepkg をインストールする例．システムに **git** (1) がなければ，**pkgsrc/devel/git** へ移動し `make install` すればよい．

5.3　basepkg の利用方法

5.3.1　インストール

basepkg のインストール方法は GitHub からアーカイブをダウンロードし展開する方法と，**pkgsrc-wip** からインストールする方法の 2 通りがある．

GitHub では `https://github.com/user340/basepkg/releases` からアーカイブをダウンロードできる．しかし basepkg は **pkgsrc** (7) の **pkgtools/pkg_install** に依存しているため，なんらかの方法で **pkgtools/pkg_install** もインストールしなければ basepkg は実行できない．したがって我々は依存関係にあるプログラムも自動的にインストールする **pkgsrc-wip** の利用を推奨している．

pkgsrc-wip からインストールするには，システムに **pkgsrc** (7) と **pkgsrc-wip** をインストールしなければならない．**pkgsrc** (7) は安定版を `ftp.netbsd.org/pub/pkgsrc/stable` からダウンロードできる．通常，**pkgsrc** (7) は**/usr/pkgsrc** に配置される．**pkgsrc-wip** は `pkgsrc.org` の Git リポジトリを **/usr/pkgsrc** の下に clone すればよい．その後，**/usr/pkgsrc/wip/basepkg** へ移動し `make install` を実行すると**/usr/pkg/share/basepkg** へ basepkg がインストールされる．図 28 は NetBSD を例にこれらのインストールをまとめたものである．

5.3.2　NetBSD ソースツリーのビルド

basepkg は NetBSD の配布物を用いてパッケージを作成する．たとえば `ftp.netbsd.org/pub/NetBSD/NetBSD-7.1/amd64/binary/sets/` から配布物をダウンロードし，それをもとにパッケージを作るよう basepkg に指定することも可能である．しかし本節では `ftp.netbsd.org/pub/NetBSD/NetBSD-7.1/source/sets/` からソースツリーをダウンロードし，**build.sh** を用いて配布物を手元の環境で作成してから basepkg を実行する手順について述べる．この手順はソースツリーを **cvs** (1) でアップデートし続けられるため，NetBSD の最新バージョンへの追従をしやすい利点がある．

NetBSD のソースツリーは通常，**/usr/src** へ配置される．またコンパイル後の生成物は**/usr/obj**

```
# ftp ftp://ftp.netbsd.org/pub/NetBSD/NetBSD-7.1/source/sets/
...
ftp> mget *.tgz
mget gnusrc.tgz [anpqy?]? a
...
ftp> bye
# ls | grep 'tgz$' | xargs -n 1 -I % tar zxf % -C /
...
# mkdir /usr/obj /usr/tools
# cd /usr/src
# ./build.sh -O ../obj -T ../tools tools
...
# ./build.sh -O ../obj -T ../tools release
...
# cd /usr/pkg/share/basepkg
# ./basepkg pkg
...
```

図 29　NetBSD で basepkg でパッケージを作成する一連の手順をまとめたもの.

へ，ビルド用プログラムは**/usr/tools** へ配置される．ソースツリーを**/usr/src** へ展開し，**/usr/obj** と**/usr/tools** ディレクトリを作成する．**/usr/src** へ移動し `build.sh tools` と `build.sh release` を実行する．その後，**/usr/pkg/share/basepkg** へ移動し，`basepkg.sh pkg` を実行すると，`/usr/pkg/share/basepkg/packages` 下にパッケージが作成される．図 29 はこの手順をまとめたものである．

5.3.3　パッケージのインストール

basepkg によって作成されたパッケージは **pkg_add** (1) を用いてシステムへインストールできる．**pkgsrc** (7) でインストールしたパッケージと区別するため，データベースを分けてインストールすることを推奨している．たとえば **base-sys-usr** パッケージをインストールするには `pkg_add -K /var/db/basepkg base-sys-usr` とする．

例外として，etc-からはじまる名前のパッケージは`-p` オプションを付け，一時ディレクトリへ中身を退避させてから **+INSTALL** スクリプトで**/etc** 以下のファイルに反映させなければならない．たとえば `pkg_add -K /var/db/basepkg -p /tmp/basepkg etc-sys-etc` のようにする．この操作は，**/etc** 以下のファイルを上書きしてしまうのを避けるためにおこなう．`-p` オプションがなければ**/etc/passwd** や**/etc/group** といった致命的なファイルが上書きされ，ユーザがログイン不可能になる．

5.4　basepkg の設計と実装

この節では basepkg の設計および実装について述べる．はじめに，basepkg のソースツリーの構造について述べる．

basepkg のソースツリーを図 30 に示した．すべての処理は **basepkg.sh** がおこない，**sets** 以下に配置さ

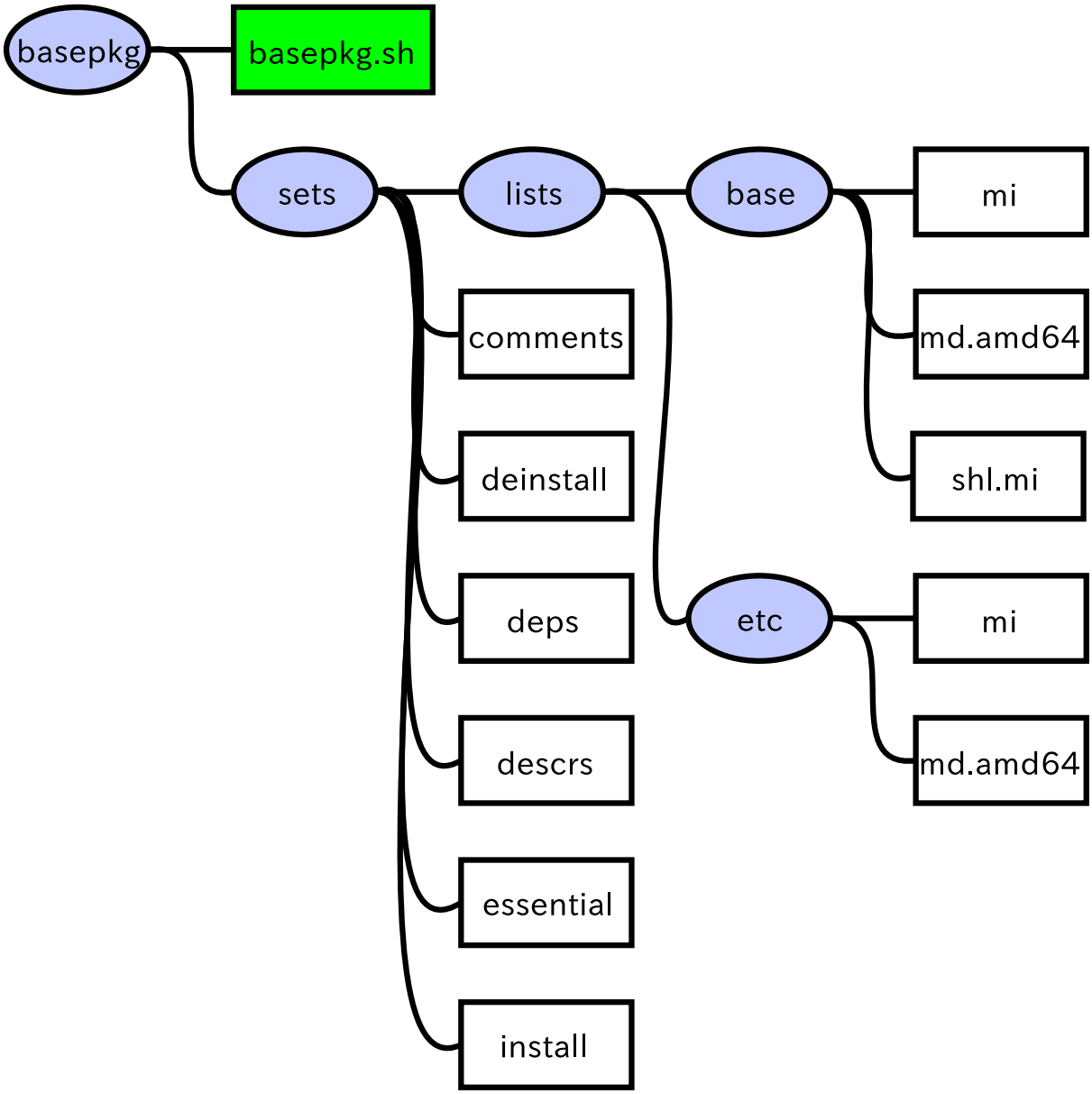


図 30　basepkg のソースツリー構造．**lists** 以下のファイルやディレクトリは一部省略した．

れている各ファイルは **basepkg.sh** から参照され，パッケージに関するデータベースとして働く．**sets** 以下のファイルは本来 NetBSD のソースツリー **usr/src/distrib/sets** に存在していたものであったが，修正が必要であったため basepkg の一部として取り込み，basepkg 独自の変更を施して利用している．

sets/lists はファイル名とパッケージ名の対応が表されたファイルが基本システムのカテゴリごとに配置されている．NetBSD の基本システムは base・comp・debug・etc・games・man・misc・modules・tests・text・xbase・xcomp・xdebug・xfont・xserver の 16 種類のカテゴリに分類されている．図 31 は **sets/lists/base/mi** の一部である．**./altroot** ディレクトリは **base-sys-root** パッケージに格納され，**./bin/cat** は **base-util-root** パッケージに格納されることを意味している．**sets/lists** 以下のファイルは

```
./altroot                                       base-sys-root
./bin                                           base-sys-root
./bin/[                                         base-util-root
./bin/cat                                       base-util-root
./bin/chgrp                                     base-util-root
./bin/chio                                      base-util-root
```

図 31　**sets/lists/base/mi** の一部．ファイル名とパッケージ名の対応が記載されている．

```
base-adosfs-root Root file system support for Amiga DOS file system support
base-amd-bin auto-mounter daemon
base-amd-examples example configuration files for the auto-mounter daemon and utilities
base-amd-shlib auto-mounter daemon shared library
base-audio-bin utilities for playing and recording audio
```

図 32　**sets/comments** の一部．第 1 項はパッケージ名で，以降はその説明が書かれている．

```
base-adosfs-root base-sys-root
base-adosfs-root base-sys-shlib
base-amd-bin base-amd-shlib
base-amd-bin base-crypto-shlib
base-amd-bin base-ldap-shlib
```

図 33　**sets/deps** の一部．パッケージの依存関係が書かれている．

すべてこのような形式のデータベースである．

Unix では各項の区切り文字に空白またはタブを用いる．したがって図 31 では 1 行目の第 1 項が “./altroot” であり，第 2 項が “base-sys-root” となる．以降登場するデータベースも同様の形式である．

sets/comments は各パッケージについての短い説明が記載されている．図 32 は **sets/comments** の一部を抜粋したものである．各行の第 1 項はパッケージ名である．第 2 項以降はそのパッケージについての短い説明が記載されている．

sets/deinstall と **sets/install** は **pkg_add** (1) と **pkg_delete** (1) が用いるスクリプトの原型である．

sets/deps にはパッケージの依存関係の情報が図 33 のように書かれている．この例では，**base-adosfs-root** パッケージが **base-sys-root** と **base-sys-shlib** パッケージに依存していることを意味する．

sets/descrs は **sets/comments** と似ているが，これより長い説明が書かれている．たとえば図 34 のように，1 行のみの **sets/comments** とは異なり複数行に渡る説明を書くことが可能である．

sets/essentials は削除不可能なパッケージの一覧である．このファイルに書かれているパッケージをシステムから削除することはできない (図 35)．

basepkg の処理は「**sets/lists** データベースの読み込み」と「メタデータの作成」・「パッケージの作成」の

```
comp-c-bin
comp-c-bin      This package includes compilers and tools for the C programming language:
comp-c-bin      c89, c99, cc, gcc - C compilers
comp-c-bin      flex, lex - lexical analyzer generator
comp-c-bin      lint - C program validator
comp-c-bin      rpcgen - RPC stub generator
comp-c-bin      yacc - parser generator
```

図 34　**sets/comments** の一部．ここでは **comp-c-bin** パッケージの説明が述べられているが，このように複数行に渡って書くことも可能である．

```
base-pkgutil-bin
base-sys-examples
base-sys-root
base-sys-share
base-sys-shlib
```

図 35　**sets/essentials** の一部．このファイルに書いてあるパッケージはシステムに必ず必要であることを意味する．

3 段階に別れる．以降は、これらについて、それぞれの小節で述べる．

5.4.1　sets/lists データベースの読み込み

basepkg は **sets/lists** 内のデータベースを読み込み，作業ディレクトリ**$workdir** にパッケージごとのディレクトリを作成する．**$workdir** とは，たとえば NetBSD 7.1 の amd64 アーキテクチャ向けパッケージを作るのであれば **work/7.1/amd64** を意味する．本稿ではこのようなバージョン・アーキテクチャ別に区切られた作業ディレクトリを**$workdir** と表現する．

はじめに，**FILES** という名の，ファイルとパッケージ名の対応データベースを**$workdir/カテゴリ名/**に作成する．この処理は `split_category_from_list ()` 関数がおこなう．**FILES** は **sets/lists** 以下のデータベースのようなものであるが，“@MODULEDIR@” や “@MACHINE@”・“@OSRELEASE@” といったバージョンやアーキテクチャによって書き換えなければいけない箇所を編集している．また “obsolete” という名前や識別子がつけられたパッケージは作成しないと syspkg では定められているため，basepkg もそれに習い obsolete なパッケージは **FILES** には書かないようにしている．図 36 は base カテゴリの **FILES** の内容の一部を抜粋したものである．

次に，この **FILES** の第 2 項をみて，パッケージと同じ名前のディレクトリを**$workdir/カテゴリ名/**へ作成する．この処理は `make_directories_of_package ()` 関数がおこなう．結果，図 37 のようなディレクトリ階層になる．

```
altroot base-sys-root
bin base-sys-root
bin/[ base-util-root
bin/cat base-util-root
bin/chgrp base-util-root
```

図 36　**$workdir/base/FILES** の内容の一部.

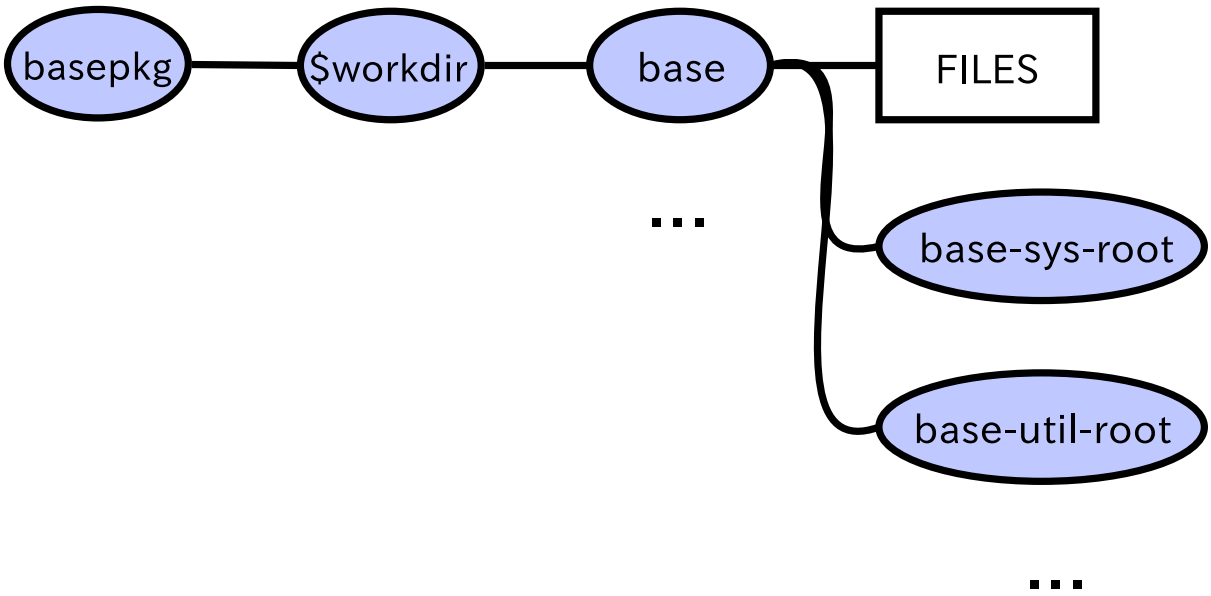


図 37　「**sets/lists** データベースの読み込み」が終了したさいのディレクトリ階層のイメージ.

```
base-sys-root var/db/obsolete/modules var/tpm var/tmp var/spool/sockets ...
base-util-root var/tmp/vi.recover var/quotas var/preserve var/msgs bin/test ...
```

図 38　**$workdir/base/CATEGORIZED** の一部．行の第 1 項にはパッケージ名，第 2 項以降はそのパッケージに格納されるファイルやディレクトリ名が続く.

5.4.2　メタデータの作成

作業ディレクトリへパッケージごとにディレクトリを区切ったあと，**sets/**以下のファイルを読み込んでパッケージのメタデータを作成する.

はじめに各パッケージの **+CONTENTS** を作成する．**FILES** の情報をもとに図 38 のように行の第 1 項がパッケージ名，第 2 項以降がそれに格納されるファイル名が続く形式のデータベース **CATEGORIZED** を作成する.

そして **CATEGORIZED** をもとに，各パッケージのディレクトリにそのパッケージが格納するファイル

```
var/tmp/vi.recover
var/quotas
var/preserve
var/msgs
bin/test
...
```

図 39　**$workdir/base/base-util-root/PLIST** の一部.

```
@name base-util-root-7.1
@comment Packaged at 2017-12-29 15:02 UTC by uki@uki-PC-VK26MCZNH
@pkgdep base-rumpclient-shlib>=7.1
@pkgdep base-sys-root>=7.1
@pkgdep base-sys-usr>=7.1
@cwd /
@exec install -d -o root -g wheel -m 01777 var/tmp/vi.recover
@exec install -d -o root -g wheel -m 0750 var/quotas
@exec install -d -o root -g wheel -m 0755 var/msgs
@exec install -d -o root -g wheel -m 0755 var/preserve
bin/[
bin/cat
bin/chio
...
```

図 40　**$workdir/base/base-util-root/+CONTENTS** の一部.

のリスト **PLIST** を作成する (図 39)．この一連の処理は `make_contents_list ()` 関数がおこなう．

PLIST をみて **+CONTENTS** が作成される．この処理は `make_CONTENTS ()` によっておこなわれる．**pkg_add** (1) が解釈する，“@” から始まる命令行が追加され，各パッケージのディレクトリへ配置される (図 40)．

ここで，パッケージの依存関係は **sets/lists/deps** を参照して計算される．この処理は `culc_deps ()` 再帰関数によっておこなわれる．この関数はパッケージ A の依存関係が以下のように記されている場合，

```
A B
B C
C D
A E
```

依存関係にあるすべてのパッケージ名を一時ファイルに出力する．

```
B
C
```

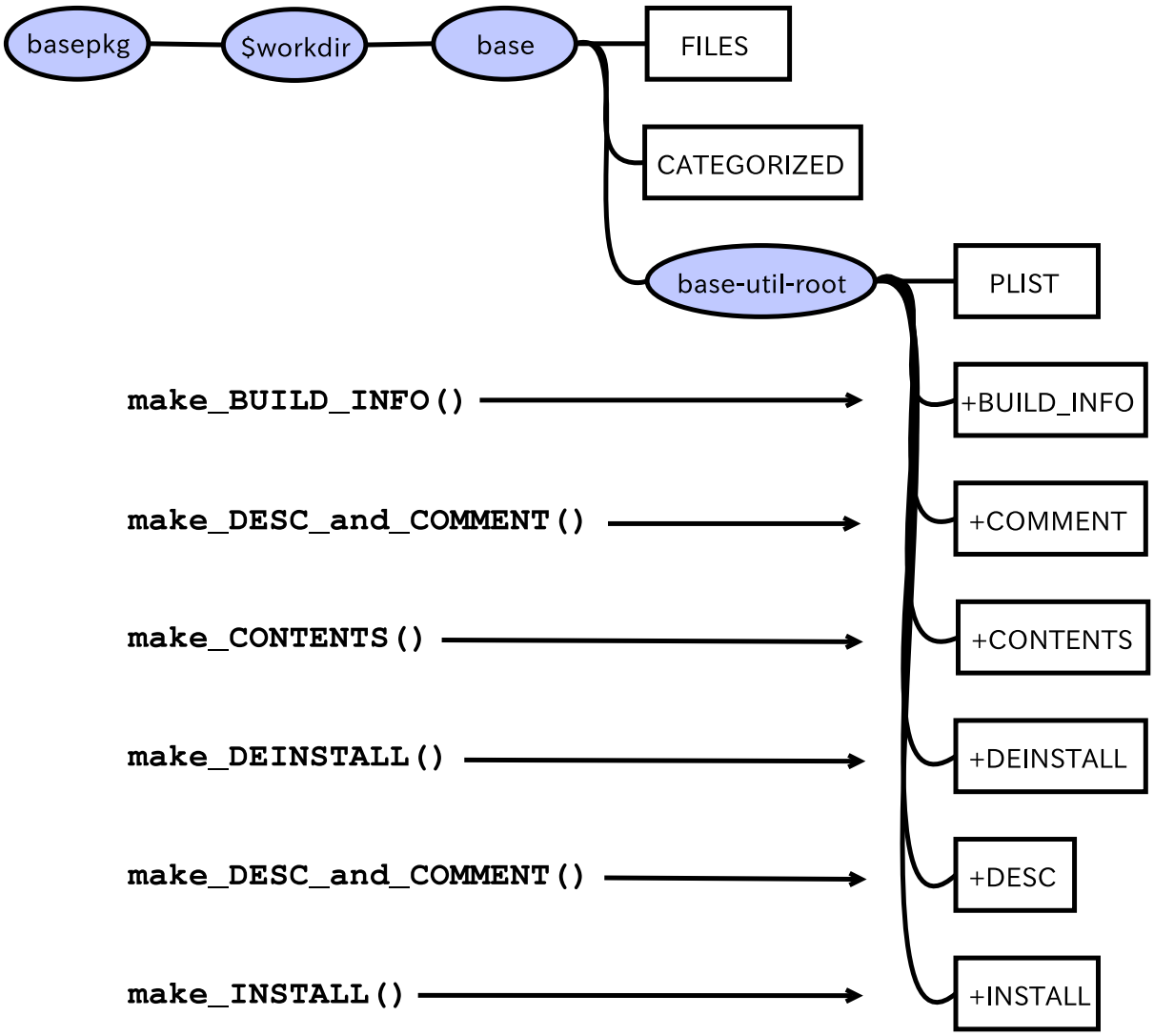


図 41　メタデータ作成後の**$workdir** ディレクトリ構造およびメタデータを作成した関数の対応.

D

E

以降は **+BUILD_INFO** · **+DESC** · **+COMMENT** · **+INSTALL** · **+DEINSTALL** といったメタデータを，**sets**/以下のデータベースを参照して作成する．この処理はそれぞれ `make_BUILD_INFO ()`・`make_DESC_and_COMMENT ()`・`make_INSTALL ()`・`make_DEINSTALL ()` 関数がおこなう．

これらの処理が終了すると，**$workdir** 以下の構造は図 41 のようになる．

5.4.3　パッケージの作成

パッケージの作成は `do_pkg_create ()` 関数によっておこなわれる．この関数は作業ディレクトリ下の各パッケージのディレクトリに存在するメタデータを参照し，**/usr/obj**/ディレクトリ以下のバイナリを格納したパッケージを作成する．

パッケージは **pkg_create** (1) にて作成され，**packages/バージョン/アーキテクチャ/**ディレクトリへ生成される．NetBSD 7.1 amd64 のパッケージを作成したのであれば，**packages/7.1/amd64-x86_64** 以下にパッケージが生成される．

6 議論

6.1 パッケージ生成手法における syspkg との比較

basepkg はシェルスクリプトのみによって書いたことで，コードは読みやすくなり，今後の保守運用性能がよくなったが，そのぶん syspkg に比べ実行時間が 3 割ほど余分に必要であり，コーディングには改善の余地が残されている．また，**pkgsrc** (7) フレームワークと syspkg 由来のデータベースを用いて syspkg にはない機能の実装もおこなったが，データベースの管理運用方法には課題が残されている．

basepkg は syspkg よりも少ないソースコード行数で実装できた．**basepkg.sh** は，バージョン 1.2 では 1190 行のシェルスクリプトである．一方で syspkg は，**usr/src/distrib/syspkg** 内に存在するすべての Makefile の行数の合計は 1008 行，**usr/src/distrib/sets/**内に存在するすべてのシェルスクリプトおよび AWK スクリプトの行数の合計は 2721 行である．したがって syspkg は合計 3729 行のスクリプト言語と Makefile から構成されている．

また basepkg のソースツリーを `find -type d | wc -l` で検索すると，ディレクトリの数が 45 個であったのに対し，syspkg の **usr/src/distrib/sets** と **usr/src/distrib/syspkg** を同様に検索すると，それぞれ 36 個と 846 個であった．したがって，basepkg は syspkg より小さなディレクトリ階層をもっていることになり，ソースツリー全体の把握をしやすい．

syspkg と異なり，basepkg は pkgsrc フレームワークを最大限に活かすことで，機能を実装した．具体的には，パッケージのインストール時にシェルスクリプトを用いてユーザやグループの追加および削除・指定パスへのファイルのコピーやディレクトリの作成・パーミッションの変更といったインストール時の制御をおこなえるようになった．

syspkg では NetBSD カーネルを格納するパッケージは作成不可能であるが，basepkg では `basepkg kern` を実行することで，**/usr/obj/sys/arch/アーキテクチャ/compile** 下に存在するカーネルをすべてパッケージ化できる．

しかし `build.sh syspkgs` と `basepkg.sh pkg` の実行速度には差がある．Intel Xeon CPU 2.40GHz, RAM 8GB の NetBSD 7.1 amd64 にてこれらのコマンドをそれぞれ 5 回ずつ実行し，実行開始と実行終了時点で **date** (1) を実行し，それらの差分をとることで測定した．**time** (1) を用いなかったのは，**build.sh** が **date** (1) で処理の開始時刻と終了時刻を記録するため，**basepkg.sh** も同様の計測方法をとったためである．

表 3 は実行時間の平均の比較である．`build.sh syspkgs` の実行時間の平均は 850.4 秒であったが，`basepkg.sh pkg` の実行時間の平均は 1188.4 秒であった．つまり basepkg は syspkg に比べ 3 割程度実行が遅いことになる．現在の basepkg はシェルスクリプトを手続き型言語のように使っているため，可能な限り松浦らが言うようなストリーミング型の記述 [65] に置き換えていくことで実行時間の改善が見込めると考えられる．

6.2 パッケージ更新速度評価におけるパッケージの依存関係数について

我々は basepkg で作成されたパッケージのインストール実時間の評価を，$_{11}C_4$ 通りまでしかおこなわなかった．これは**$workdir/7.1/amd64** 以下の **+CONTENTS** に記述されている “@pkgdep” の行数を各パッケージごとに求めた結果，パッケージの依存数は 3 つ以下が 8 割以上を占めていたためである．

すなわち，ユーザが実際に特定のパッケージをアップデートするさいは同時にアップデートされるパッケー

表 3　`build.sh syspkgs` と `basepkg.sh pkg` の実行時間の平均.

プログラム	実行時間 (秒)
`build.sh syspkgs`	850.4
`basepkg.sh pkg`	1188.4

```
{
    if ($1 in list)
        list[$1] = $2 " " list[$1]
    else
        list[$1] = $2
}
END {
    for (pkg in list)
        print pkg, list[pkg]
}
```

図 42　**grep** (1) の出力を入力とし，パッケージとその依存パッケージの対応を 1 行にして出力する **read.awk**

ジの数がおおよそ 3 つ以下，つまり合計 4 つであることを考えると，${}_{11}C_4$ 通りまでの計測結果を根拠に，配布物をダウンロードして展開するよりもパッケージによるインストールまたはアップデート速度のほうが速いと言える.

パッケージの依存数の確認には図 42 と図 43，図 44 にある 3 種類の AWK スクリプトを使用し，以下のように実行する.

```
$ grep "@pkgdep" /usr/pkg/share/basepkg/work/7.1/amd64/*/*/+CONTENTS | \
? awk -f read.awk | awk -f count.awk | awk -f depnum.awk
1 92
2 240
3 421
4 75
5 8
6 2
7 6
8 3
```

この出力は，行の第 1 項がパッケージの依存数を，第 2 項がその依存数であるパッケージの数を表している.

```
{
    print gsub(/ /, " ", $0)
}
```

図 43　**read.awk** の出力を入力とし，パッケージの依存数を出力する **count.awk**

```
{
    if ($1 in list)
        list[$1] += 1
    else
        list[$1] = 1
}
END {
    for (i in list)
        print i, list[i]
}
```

図 44　**count.awk** の出力を入力とし，パッケージの依存数の分布を出力する **depnum.awk**

6.3　今後の課題

6.3.1　安全な **pkg_add** (1) インタフェース

pkg_add (1) を basepkg のパッケージに用いるには，オプションの指定やインストール後のデーモンの管理といった作業がユーザに求められる．これらの作業を自動でおこなう **pkg_add** (1) のインタフェースがあればユーザにとって便利である．

現在，basepkg によって作成されたパッケージのインストールは，ユーザ自身が **pkg_add** (1) にオプションを指定して実行しなければならない．たとえば **pkgsrc** (7) のシステムデータベースと区別するために`-K` オプションは必要となる．しかし，これは特に **etc-sys-etc** パッケージのような etc カテゴリのパッケージをインストールするさいに致命的なトラブルを引き起こす可能性が高い．第 5.3.3 節で述べたように，etc カテゴリのパッケージをインストールするには **pkg_add** (1) の`-p` オプションを用いなければ，既存の**/etc/group** や **/etc/passwd** を上書きしてしまう．このようなトラブルを未然に防ぐため，basepkg パッケージ用の **pkg_add** (1) インタフェースが必要となる．たとえば，図 45 のようなスクリプトになるだろう．

このように，ユーザが安全に basepkg によるパッケージを扱えるよう，可能な限りシステムの安全に配慮したインタフェースが求められる．

またデーモンプログラムのパッケージをアップデートしたさいに自動でそのデーモンを再起動しなければ，古いデーモンがシステム中で動き続けることになり，アップデートの意味が薄れてしまう．特定のパッケージの **+INSTALL** にデーモンを再起動させる処理を加えるなどの実装が必要である．

```
#!/bin/sh

option="-K /var/db/basepkg"
test "${1%%-*}" = "etc" && option = "-p /var/cache/basepkg $option"

pkg_add "$option" "$1"
```

図 45　**pkg_add** (1) インタフェースの例．インストール対象が etc カテゴリのパッケージであった場合，-p オプションを付与させる．

6.3.2　basepkg によるパッケージの配布

Debian や RHEL のような Linux ディストリビューションは，ビルドされたパッケージをサーバ上で配布している．日本では **http://ftp.jp.debian.org/debian/**がある．また FreeBSD の **pkg** (8) も，**ports** (8) を使わない場合はネットワーク上のパッケージリポジトリからパッケージをダウンロードし，インストールをおこなう．

パッケージによるシステムの管理は，作成されたパッケージをネットワーク上から見つけ，それをシステムにインストールするのが一般的である．また高速なパッケージ作成サーバを構築し，パッケージを配布することによってユーザがソースツリーをビルドする作業を省略でき，利便性が高まる．

そこで我々は basepkg を用いたパッケージの配布サーバを運用している．しかし NetBSD 特有の対応アーキテクチャ数やソースツリーのビルド時間，パッケージの作成時間を検討すると，現状のシステムではセキュリティアップデートのような緊急時に迅速に脆弱性対応済みのパッケージを配布できないことがわかった．このサーバの運用や評価，課題など，詳細は付録 E で述べる．

6.3.3　basepkg データベースの保守・管理

basepkg は **usr/src/distrib/sets** に存在する syspkg のデータベースに修正を加えて利用しているが，NetBSD のバージョンアップデートのさいにどのようにそのアップデートに追従し，内容の正当性を保証するかが問題となる．特に **basepkg/sets/lists/**内のファイルは，パッケージに格納されるファイルの情報であり，アップデートの追従に関して確実に変更が反映されていなければならない．

現在の basepkg のパッケージは NetBSD 7.1 の構成要素をすべてパッケージ化できていることは確認しているが，syspkg がどのように **usr/src/distrib/sets/**以下をアップデートし続けてきたのか不明であるため，NetBSD 8.0 がリリースされるまでに検討をおこなわなければならない．

また **basepkg/sets/comment** と **basepkg/sets/descr** は内容が不完全である．これらはパッケージの説明を記したデータベースであり，basepkg や **pkg_add** (1) の動作に直接の関係があるわけではないものの，ユーザの利便性を考慮するとこれらのデータベースの内容を完全にする作業にも着手すべきである．

6.3.4　パッケージの命名について

basepkg によるパッケージの名前は syspkg のものを引き継いでいる．ディレクトリ階層をまとめたパッケージであれば “root”，バイナリをまとめたパッケージであれば “bin”……と，ある一定の命名規則に従って

パッケージの名前が命名されている．

しかしこの名前づけはユーザにとって不親切なものであることが判明した．たとえば，セキュアなリモートログインに用いられる OpenSSH クライアントプログラム **ssh** (1) は **base-secsh-bin** パッケージの内容物であるが，“secsh” という名前から OpenSSH を連想することは困難である．OpenSSH の開発プロジェクトは “openssh” という名前でプログラムをリリースしている．また **pkgsrc** (7) においても **security/openssh** に登録されていることから，**base-secsh-bin** を **base-openssh-bin** に変更すると，よりユーザにとって理解しやすいパッケージ名となるだろう．

しかし一方でパッケージの名前を変更するだけでは解決しない場合もある．暗号プロトコルを実装したライブラリである OpenSSL の共有ライブラリ **libssl.so** は，**base-crypto-shlib** パッケージの内容物であるが，同じく OpenSSL のライブラリである **libssl.a** は **comp-c-lib** パッケージの内容物である．このようにひとつのライブラリに関するファイルが複数のパッケージに分散されて格納されている場合もあるため，各パッケージの内容物を一度すべて検査し直す必要があるだろう．これは時間がかかり困難な作業である．

6.3.5　その他システム管理プログラムとの連携

ユーザにとって，パッケージの脆弱性情報を自動で入手できると利便性が高まる．basepkg は **pkgsrc** (7) のパッケージ管理システムを用いてパッケージを作成しているため，**pkgsrc** (7) の脆弱性対応フレームワークとの連携が理論上可能である．

第 3.3.5 節で述べたように，**pkgsrc** (7) は **pkg_admin** (1) を用いて，NetBSD Project 公式の FTP サーバから脆弱性情報の一覧をダウンロードできる．これと同じく basepkg も pkg-vulnerabilities と同じ形式の脆弱性情報のデータベースを配布サーバに用意することで，脆弱性情報の取得を自動でおこなえるようになるだろう．

また NetBSD の基本システムにはまだ組み込まれていないが，FreeBSD の **pkg** (8) のようにパッケージを特定のリポジトリから探し，自動的にパッケージをインストールする NetBSD 用第三者製ソフトウェア **pkgtools/pkgin** が存在する．**/var/db/pkg** と **/var/db/basepkg** の区分の問題や，etc カテゴリパッケージのインストールの問題など課題は残るものの，basepkg によるパッケージと **pkgtools/pkgin** との連携がおこなえるようになれば **pkgtools/pkgin** ひとつでシステムすべてのパッケージを管理できるようになる．

7　結論

BSD は基本システムがパッケージで管理されておらず，Debian や RHEL のようにシステム全体をパッケージで管理しているディストリビューションに比べ基本システムのアップデートに時間がかかっていた．また基本システムがカテゴリ単位で分割されているため，基本システム内の一部のプログラムを更新するには，ひとつのカテゴリごとアップデートしなければならず非効率的であった．

各 BSD の開発プロジェクトでは基本システムをパッケージに分割する機能を開発しているが，特に NetBSD はその機能を提供する syspkg の開発が停滞していた．

そこで我々は，NetBSD 基本システムをパッケージ化するシステム basepkg を開発した．

syspkg は「Makefile + 複数のシェルスクリプト + 複数のディレクトリ + データベース」というシステム構成であったが，basepkg は「単一のシェルスクリプト + データベース」という構成になっており，syspkg と比べコンパクトに開発できた．

また **pkg_add** (1) や **pkg_delete** (1) のさいに実行されるインストール制御用スクリプトの実装や，データベースの修正など，syspkg より優れた細かな粒度での効率的な基本システム管理のための機能を実装した．

また，basepkg によるパッケージ配布サーバも試験運用している．パッケージ配布サーバによって，ユーザ自身でソースツリーをビルドする必要がなくなり，高速な基本システムのアップデートが実現可能となる．

本システムは NetBSD ユーザにとっての利便性向上に寄与すると考えられるが，解決するべき課題も多いことが分かった．basepkg のようなシステムの基幹に直接影響を及ぼすプログラムは，開発しただけでは不具合の発見が難しく，複数人による長期の試験運用によって実装すべき機能や修正すべき不具合が発見されるためである．

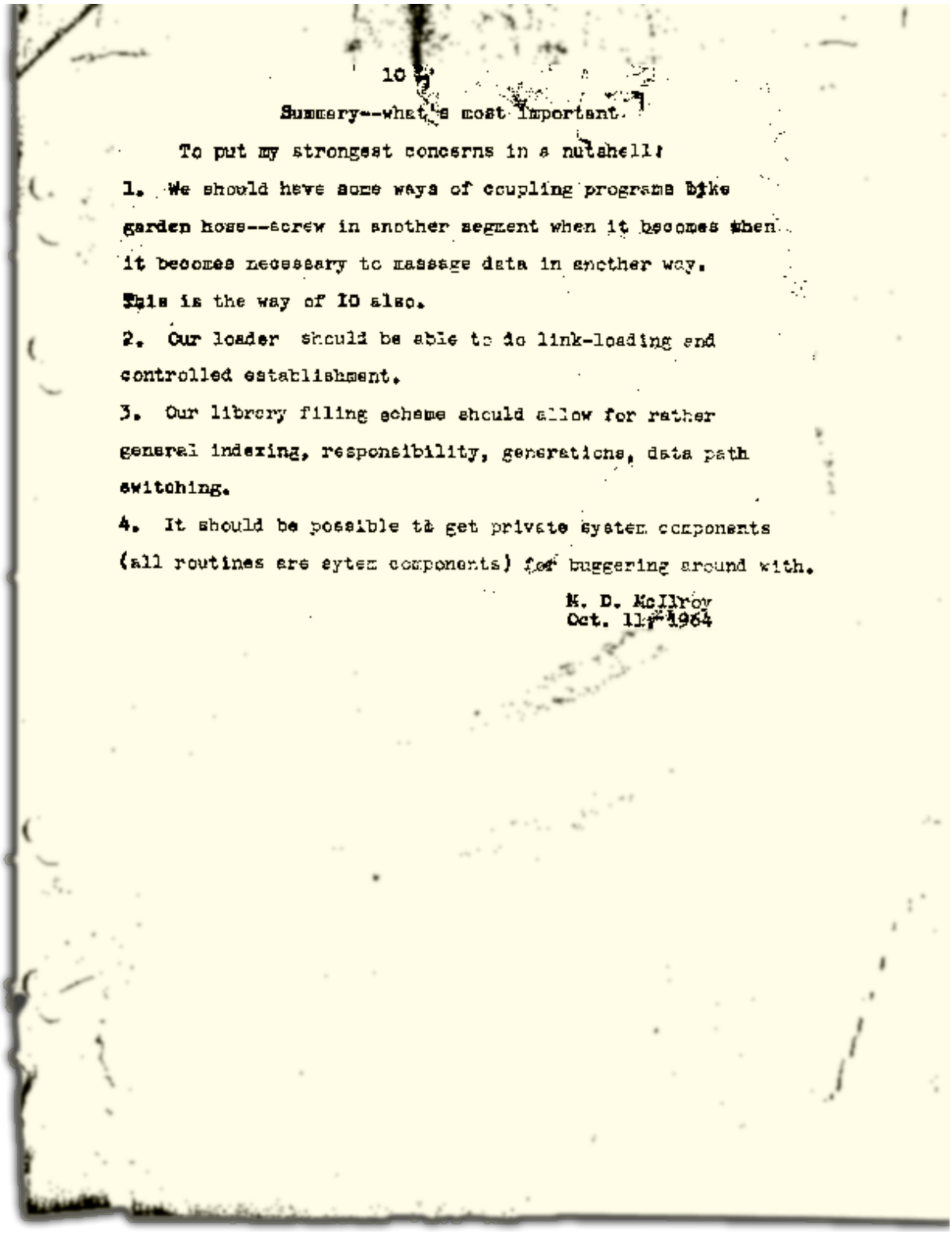
本研究により既存の手法以上の機能を備えたプログラムを実装できた．また試験運用をおこなっているが，解決すべき課題も見つかっている．

謝辞

論文添削やプログラムの実装についてご指導頂いた深町賢一先生，主査の小松川浩先生，また副査の吉田淳一先生，信州大学の不破泰先生に感謝申し上げます．

また 2017 年 7 月 8 日の日本 NetBSD ユーザーグループ第十九回定期総会および NetBSD BOF 2017 では，日本 NetBSD ユーザーグループ様から交通費支援をしていただき，また発表の機会もいただきました．深く感謝申し上げます．

付録 A Doug McIlroy によるパイプのアイデアのスキャン画像 [68]



10

Summary--what's most important.

To put my strongest concerns in a nutshell:

1. We should have some ways of coupling programs like garden hose--screw in another segment when it becomes when it becomes necessary to massage data in another way. This is the way of IO also.

2. Our loader should be able to do link-loading and controlled establishment.

3. Our library filing scheme should allow for rather general indexing, responsibility, generations, data path switching.

4. It should be possible to get private system components (all routines are sytem components) for buggering around with.

M. D. McIlroy
Oct. 11, 1964

付録 B 1974 年の Unix ライセンスのコピー (Ritche のホームページ [22] より)

SOFTWARE AGREEMENT

between

WESTERN ELECTRIC COMPANY, INCORPORATED

and

KATHOLIEKE UNIVERSITEIT

Effective as of

December 1, 1974

copie UNIX-V5/V6

(SC)
Software-Univ.–020173-060574–1, Alt.

SOFTWARE AGREEMENT

Effective as of December 1, 1974 ,
WESTERN ELECTRIC COMPANY, INCORPORATED, a New York corporation ("WESTERN"), having an office at 222 Broadway, New York, New York 10038, and KATHOLIEKE UNIVERSITEIT

("LICENSEE"), having an office at Nijmegen, The Netherlands,

agree as follows:

ARTICLE I

DEFINITIONS

1.01 Terms in this agreement (other than names of parties and Article headings) which are in capital letters shall have the meanings specified in the Definitions Appendix.

ARTICLE II

GRANTS OF RIGHTS TO USE LICENSED SOFTWARE

2.01 WESTERN grants royalty-free to LICENSEE, a personal, nontransferable and nonexclusive right to use the LICENSED SOFTWARE, solely for academic and educational purposes and solely at the following location of the LICENSEE:

Computer Graphics Group
Faculteit der Wiskunde en Natuurwetenschappen
Toernooiveld
Nijmegen
The Netherlands

2.02 LICENSEE shall pay to WESTERN, upon execution of this agreement by LICENSEE, a service charge in the amount of One Hundred Fifty ---------- United States dollars ($150.00--------) for the cost of furnishing the LICENSED SOFTWARE. Delivery of the LICENSED SOFTWARE to LICENSEE shall be within a reasonable time after payment of such service charge.

(F)
Software-Univ.–020173-122073–1(a), Alt.

2.03 LICENSEE hereby assures WESTERN that it does not intend to and will not knowingly, without the prior written consent of the Office of Export Administration of the U.S. Department of Commerce, Washington, D.C. 20230, transmit directly or indirectly:

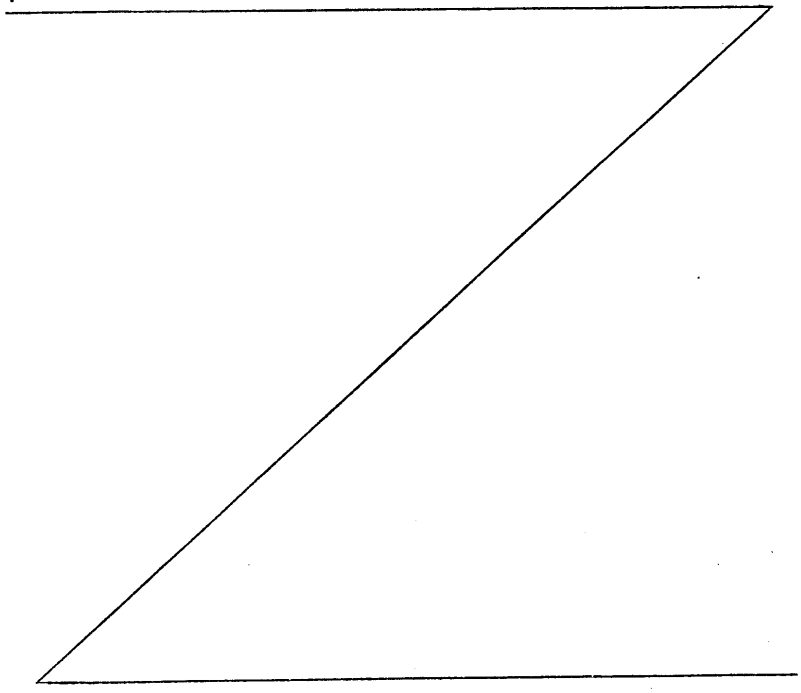
(i) any immediate product (including processes and services) produced directly by the use of the LICENSED SOFTWARE; or

(ii) any commodity produced by such immediate product if the immediate product of the LICENSED SOFTWARE is a plant capable of producing a commodity or is a major component of such plant;

to any Q, W, Y or Z country specified in Supplement No. 1 to Section 370 of the Export Administration Regulations issued by the U.S. Department of Commerce.

LICENSEE agrees to promptly inform WESTERN in writing of any such written consent issued by the Office of Export Administration.

LICENSEE hereby assures WESTERN that it will not transmit, directly or indirectly, the LICENSED SOFTWARE or any portion thereof, without the prior written consent of WESTERN, to any Q, W, Y or Z country.



ARTICLE III

TERMINATION

3.01 If LICENSEE shall fail to fulfill one or more of its obligations under this agreement, WESTERN may, upon its election and in addition to any other remedies that it may have, terminate the rights granted hereunder at any time; upon such termination LICENSEE shall within thirty (30) days deliver to WESTERN all documentation containing the LICENSED SOFTWARE, and shall render unusable all LICENSED SOFTWARE placed in any storage apparatus.

ARTICLE IV

MISCELLANEOUS PROVISIONS

4.01 Nothing contained herein shall be construed as conferring by implication, estoppel or otherwise any license or right under any patent or trademark, whether or not the exercise of any right herein granted necessarily employs an invention of any existing or later issued patent.

4.02 This agreement shall prevail notwithstanding any conflicting terms or legends which may appear in the LICENSED SOFTWARE.

4.03 WESTERN and its ASSOCIATED COMPANIES make no representations or warranties, expressly or impliedly. By way of example but not of limitation, WESTERN and its ASSOCIATED COMPANIES make no representations or warranties of merchantability or fitness for any particular purpose, or that the use of the LICENSED SOFTWARE will not infringe any patent, copyright or trademark of any third person. WESTERN and its ASSOCIATED COMPANIES shall not be held to any liability with respect to any claim by LICENSEE or a third party on account of, or arising from, the use of such LICENSED SOFTWARE.

4.04 Neither the execution of this agreement nor anything in it or in the LICENSED SOFTWARE shall be construed as an obligation upon WESTERN or any of its ASSOCIATED COMPANIES to furnish any person, including the LICENSEE, any assistance of any kind whatsoever, or any information or documentation other than the LICENSED SOFTWARE.

4.05 LICENSEE agrees that it shall hold the LICENSED SOFTWARE in confidence for WESTERN and its ASSOCIATED COMPANIES. LICENSEE further agrees that it shall not make any disclosure of the LICENSED SOFTWARE or any portion thereof (including methods or concepts utilized therein) to anyone, except to employees or students of the LICENSEE to whom such disclosure is necessary to the use for which rights are granted hereunder. LICENSEE shall appropriately notify each employee and student to whom any such disclosure is made that such disclosure is made in confidence and shall be kept in confidence by him.

4.06 The obligation of the LICENSEE and of its employees and students under Section 4.05 shall survive and continue after any termination of rights under this agreement; however, such obligations **shall not extend** to any information or technical data relating to the LICENSED SOFTWARE which is now available to the general public or which later becomes available to the general public by acts not attributable to LICENSEE, its employees or students.

4.07 LICENSEE agrees that it will not make or have made, or permit to be made, any copies of the LICENSED SOFTWARE or portions thereof, except those copies which are necessary to the use, by the LICENSEE, for which rights are granted hereunder, that each such necessary copy shall contain the same proprietary notices or legends which appear on the LICENSED SOFTWARE or which are applicable to such portions thereof, and that no rights are granted under this agreement expressly or impliedly with respect to any copyrights except as provided for in this Section 4.07.

4.08 Neither this agreement nor any rights hereunder, in whole or in part, shall be assignable or otherwise transferable.

4.09 Nothing in this agreement grants to LICENSEE the right to sell, lease or otherwise transfer or dispose of the LICENSED SOFTWARE in whole or in part.

4.10 Any notice hereunder shall be deemed to be sufficiently given and any delivery hereunder deemed made when sent by registered mail addressed to the addressee at its office above specified or at such changed address as the addressee shall have specified by written notice.

4.11 This agreement sets forth the entire agreement and understanding between the parties as to the subject matter hereof and merges all prior discussions between them, and neither of the parties shall be bound by any conditions, definitions, warranties, understandings or representations with respect to such subject matter other than as expressly provided herein, or in any prior existing written agreement between the parties, or as duly set forth on or subsequent to the effective date hereof in writing and signed by a proper and duly authorized representative of the party to be bound thereby.

4.12 The construction and performance of this agreement shall be governed by the law of the State of New York.

IN WITNESS WHEREOF, each of the parties has caused this agreement to be executed in duplicate originals by its duly authorized representatives on the respective dates entered below.

WESTERN ELECTRIC COMPANY, INCORPORATED

By .....
Director of Patent Licensing

Date February 26, 1975

Attest:

[SEAL]

.....
Secretary

.....KATHOLIEKE.UNIVERSITEIT.....

By .....
(Dr. C.J.M. Aarts)

Title Director.Faculty.of.Science.....

Date February.6,.1975.....

Attest:

[SEAL]

.....
Secretary

5

DEFINITIONS APPENDIX

ASSOCIATED COMPANIES of WESTERN means SUBSIDIARIES of WESTERN, companies presently having WESTERN as a SUBSIDIARY and other SUBSIDIARIES of such companies and their SUBSIDIARIES. The term also means and includes Cincinnati Bell Inc., an Ohio corporation, and The Southern New England Telephone Company, a Connecticut corporation, and their SUBSIDIARIES.

SUBSIDIARY means a company the majority of whose stock entitled to vote for election of directors is now or hereafter controlled by the parent company either directly or indirectly, but any such company shall be deemed to be a SUBSIDIARY only so long as such control exists.

LICENSED SOFTWARE means the computer programs, or any portions thereof, and the associated documentation therefor, listed below:

UNIX time shared operating system computer program and supporting programs and documentation consisting of the following:

(a) a first reel of 9 track 800 bpi magnetic tape supplied by LICENSEE in required length on which the following will be recorded:

(1) Binary object code of the UNIX core resident operating system;

(2) Source program code;

(3) User's manual.

(b) a second reel of 9 track 800 bpi magnetic tape supplied by LICENSEE in required length on which the binary object code of the supporting programs will be recorded.

(c) printed documentation:

(1) UNIX programmer's manual;

(2) Bell Telephone Laboratories, Inc. technical memorandum entitled NROFF User's Manual;

(3) C reference manual;

(4) UNIX assembler reference manual.

付録 C 1983 年の Unix ライセンス料金表 (Ritche のホームページ [22] より)

Software List

for

UNIX* System V

September 1, 1983

Version	Distribution Fee
UNIX System V, 11/780 Version (Also runs on 11/750)	$400
UNIX System V, 11/70 Version	$400
UNIX System III, 11/780 Version	$400
UNIX System III, 11/70 Version (Also runs on 11/45)	$400
UNIX System III, 11/44 Version	$400
UNIX System III, 11/34 Version	$400
UNIX System III, 11/23 Version (Distributed on 2 RL02 Disks)	$400
UNIX 32V Time-Sharing System, Version 1.0 (Runs on 11/780)	$400
UNIX Time-Sharing System, Seventh Edition (Runs on 11/45, 11/70)	$400
UNIX Time-Sharing System, Sixth Edition (Runs on 11/34, 11/40, 11/45, 11/70)	NA #
UNIX Programmer's Workbench System, Edition 1.0 (Runs on 11/45, 11/70)	$400
UNIX Mini Time-Sharing System, Version 6 (Runs on 11/10, 11/20, 11/34, 11/40)	NA #

* UNIX is a trademark of Bell Laboratories.

\# No longer available from Western Electric or its affiliates.

付録 D 1984 年の Unix ライセンス料金表 (Ritche のホームページ [22] より)

2/24/84

AT&T INTERNATIONAL

UNIX* SYSTEM V AND ADD ON APPLICATIONS PRICES

Item	Commercial	Educational
UNIX System V, Release 2.0, Source Code (1)	$43,000.00	$800.00
Each Additional CPU	$16,000.00	$400.00
Customer Provisions Supplemental Agreement	$25,000.00	-
Multiple Source Licenses	1-32 Users $1000 1-64 Users $3500 64+ Users $7000	-
UNIX System V, Release 2.0 Administrative-Educational	-	$16,000.00
Upgrade Fees from UNIX System V, Release 1.0 (1) (2)	$2,500.00 All CPU's	$800.00 All CPU's
Add on packages (source code only.) BASIC Interpreter. First CPU.	$5,000.00	$2,500.00
BASIC Interpreter Additional CPU's	$2,500.00	$1,250.00
Customer Provisions Fee	$10,000.00	-
MC68000 Software Generation System. First CPU	$7,500.00	$3,000.00
MC68000 Software Generation System. Additional CPUs	$3,750.00	1,500.00
Customer Provisions Fee	$10,000.00	-

*UNIX is a trademark of AT&T Bell Laboratories

(1) Includes 3 months of free Level I Support

(2) Includes UNIX System V & Documenter's Workbench. Available at no cost to only those customers who upgrade to UNIX System V, Release 2.0.

Item	Commercial	Educational
UNIX Documenter's Workbench**, First CPU	$4,000.00	$1,500.00
UNIX Documenter's Workbench, Additional CPU's	$2,000.00	$500.00
Customer Provisions Fee	$3,000.00	-
UNIX Writer's Workbench** First CPU	$4,000.00	$2,000.00
UNIX Writer's Workbench Additional CPU's	$1,600.00	$800.00
Customer Provisions Fee	$3,000.00	-
"S" Statistical Software	$8,000.00	$400.00
"S" Statistical Software Additional CPU's	$3,000.00	$400.00
Customer Provisions Fee	$5,000.00	-
UNIX Instructional Workbench** (Binary). First CPU	$2,500.00	$1,250.00
UNIX Instructional Workbench (Binary). Additional CPU's	$2,500.00	$1,250.00

Sublicensing Fee Schedules

	UNIX System V	WWB	DWB	MC68000 SGS	BASIC	"S" Statistical
Up-Front Fee	$25,000	$3,000	$3,000	$10,000	$10,000	$5,000
Binary Sublicensing Fees						
1-2 User System	60	50	10	50	50	100
1-8 User System	125	100	15	100	100	200
1-16 User System	500	150	30	250	250	400
1-32 User System	1000	250	45	500	500	800
1-64 User System	3500	350	125	1500	1000	1400
64+ User System	7000	550	250	2500	1500	2000

Discounts of 2% for each $100,000 of sales commitment may be taken in sublicensing fees up to a maximum discount of $3 million or 60%.

**Documenter's Workbench, Instructional Workbench and Writer's Workbench are trademarks of AT&T Technologies, Inc.

Teletype*** 5620 Terminals

Item	Commercial	Educational
Core Package Source	$5,000.00	$5,000.00
Core Package Binary Code	$1,600.00	$1,600.00
Core Package Right-to-Copy	$800.00	$800.00
Development Package Source	$15,000.00	$15,000.00
Development Package Binary Core	$6,000.00	$6,000.00
Development Package Right-to-Copy	$3,000.00	$3,000.00
Text Processing Package Source	$3,000.00	$3,000.00
Text Processing Package Binary Code	$1,000.00	$1,000.00
Text Processing Package Right-to-Copy	$500.00	$500.00

C Language Compilers

Commercial

	First CPU	Additional CPU's	Sublicensing Fees
C/370	$4,000.00	$2,000.00	$200.00
C/SEL	$4,000.00	$2,000.00	$200.00
C/6000	$4,000.00	$2,000.00	$200.00
C/Data	$5,000.00	$2,500.00	$200.00
C/Cray	$5,000.00	$2,500.00	$200.00
Fortran 77	$3,000.00	$1,500.00	$200.00

Educational - Educational Fee $400 per CPU for all of the above except Fortran which is $400 per Agreement.

Other Commercial

Phototypesetter Programmer's Workbench	$3,000.00	$1,500.00	$200.00
Phototypesetter V7	$3,300.00	$1,100.00	$200.00
Independent TROFF	$4,000.00	$2,000.00	$200.00

Other Educational

Phototypesetter Programmer's Workbench	$400.00	$200.00	-
Phototypesetter V7	$400.00	$200.00	-
Independent TROFF	$400.00	$200.00	-

Other Educational - Administrative

Phototypesetter Programmer's Workbench	$2,000.00	$700.00	-
Phototypesetter V7	$2,000.00	$700.00	-
Independent TROFF	$2,000.00	$700.00	-

***Teletype is a trademark of AT&T Teletype

付録 E パッケージ作成・配布サーバの評価について

我々は basepkg を用いてパッケージを作成し配布する `basepkg.netbsd.fml.org` を構築し，パッケージの作成・配布を試験的におこなっている．このサーバはさくらインターネット社の「さくらの VPS」[69] 仮想 3 コア・RAM 2GB プランを契約し運用している．パッケージは NetBSD 7.1 の 30 アーキテクチャ分を提供している [70].

運用評価

パッケージの作成・配布サーバでは，NetBSD のバージョンアップやセキュリティアップデートによってソースツリーに変更が加わった場合，ユーザがパッケージを用いてシステムをアップデートできるように，可能な限り速くソースツリーをビルドし直しパッケージを作成し配布する必要がある．

配布サーバの拡張をおこなう場合，マシン性能や契約台数を決めるための基準を定めるために，どの程度の性能のマシンがどのくらいの時間をかけてソースツリーのアップデートからビルド・パッケージの作成を終えることができるのかを測らなければならない．

そこで，はじめに `basepkg.netbsd.fml.org` がパッケージを作成する時間を計測した．ソースツリーをアップデート・ビルドし，パッケージを作成する手順は以下の通りである．

`cvs update` によるソースツリーのアップデート　NetBSD のソースツリーは CVS で管理されている．ユーザプログラム **cvs** (1) を通じてアップデートやログの表示などをおこなう．`cvs update` は NetBSD Project の CVS リポジトリを見て，自環境の CVS リポジトリのアップデートをおこなうことができる．

`build.sh -u release` でソースツリーの再ビルドをおこなう　**build.sh** は通常，作業ディレクトリ下の生成物を削除してからソースツリーのビルドをおこなう．しかし **build.sh** に`-u` オプションを渡すと，作業ディレクトリ下の生成物を削除せず，ソースツリー内の変更があったファイルのみを対象にビルドをおこなう．これは一度 NetBSD ソースツリーをビルドしている必要があるが，パッケージの配布サーバであれば以前ビルドしたソースツリーを持っているため問題はない．

`basepkg.sh pkg`　これは **basepkg.sh** を用いてパッケージの作成をおこなう．

各命令とその処理時間を表 4 に示す．この処理時間は CPU1 コアのみを用いたものであるが，ひとつの

表 4　ソースツリーのアップデートからパッケージの作成までの各命令およびその処理時間.

命令	処理時間 (分)
`cvs update`	4
`build.sh -u release`	20
`basepkg.sh pkg`	20

アーキテクチャにつきおよそ 44 分かかるとわかった．現在の `basepkg.netbsd.fml.org` では 30 アーキテクチャのパッケージを提供しているため，合計 $30 \times 40 + 4 = 1204$ 分，20 時間と 4 分程度かかる計算である．

パッケージ配布サーバに求められる作成時間とは

現在の basepkg.netbsd.fml.org は 20 時間かけてひとつのバージョンの 30 アーキテクチャ分のパッケージを作成し，配布している．しかし理想としては，パッケージの提供範囲を広げ 58 アーキテクチャすべてのパッケージを，現行バージョンとそのひとつ前のバージョンのものまで提供することが望ましい．

ここで，ユーザが NetBSD のソースツリーをビルドし，セキュリティアップデートをおこなう場面を想定する．マシン性能はさくらの VPS 仮想 3 コア RAM2GB と同程度のものとする．この性能の VPS では，NetBSD のソースツリーをダウンロード・展開し，ビルドを終了させるまでにおおよそ 5 時間かかる．パッケージの作成と配布に 5 時間以上必要なのであれば，ソースツリーをダウンロードしてビルドしたほうが速く，パッケージを用いる必要はない．

したがって，パッケージを配布するサーバは 5 時間以内に全アーキテクチャのパッケージを作成し，配布しなければならないことになる．ソースツリーのアップデートにかかる時間は 4 分，変更箇所をビルドし直し，パッケージを作成するまでには 40 分かかることを考えると，2 つのバージョンの NetBSD ソースツリーをアップデートし，全アーキテクチャの変更箇所をビルドし直し，全アーキテクチャごとのパッケージを作成する作業は，現在の basepkg.netbsd.fml.org では $58(\text{アーキテクチャ数}) \times 2(\text{バージョン数}) \times 40(\text{分}) + 4(\text{分}) \times 2(\text{バージョン}) = 4648(\text{分}) \sim 77.5(\text{時間})$ かかる．

1 コアあたりひとつの **build.sh** と **basepkg.sh** を実行できるため，16 コア以上あれば処理を並行し，5 時間で完了する計算である．すなわち，単純に計算すると 6 台以上のさくらの VPS 仮想 3 コア RAM2GB があれば 5 時間でパッケージを提供可能である．この性能のプランであれば，年間契約で 18770 円かかるため，$18770(\text{円}) \times 6(\text{台}) = 112620(\text{円})$ で理想的なパッケージ配布サーバを運用可能である．

パッケージの作成・配布サーバに求められる性能やコア数を計算したが，個人による非営利的な活動に年間約 11 万円をかけるのは，経済的に継続が困難である．この結論は単純なひとつの指標である．実際の運用では，配布サーバを一台 VPS として契約し，パッケージを作成するサーバはクラウドコンピューティングを用いて時間単位で CPU とストレージを購入する運用体制になるだろう．

しかしこの運用は企業のサービスを用いるものである．また最も多いコア数のクラウドを契約しても，そのマシン上でパッケージ作成にどれほどの時間がかかるかは実際に運用してみなければ不明である．これらの理由から，年間あたり必要な具体的な金額を算出することが難しく，本稿ではこれ以上の言及ができない．

参考文献

[1] NetApplications.com. Operating system market share. net.marketshare.com/operating-system-market-share.aspx.

[2] Trend Micro. 2016 年上半期セキュリティラウンドアップ. www.trendmicro.com/ja_jp/security-intelligence/research-reports/sr/sr-2016h1.html, 2016.

[3] Trend Micro. 2017 年第 1 四半期セキュリティラウンドアップ. www.trendmicro.com/ja_jp/security-intelligence/research-reports/sr/sr-2017q1.html, 2017.

[4] CVE Details. Linux linux kernel : Cve security vulnerabilities, versions & detailed reports. www.cvedetails.com/product/47/Linux-Linux-Kernel.html?vendor_id=33.

[5] Steven Levy. ハッカーズ. 工学社, 1987.

[6] Eric S. Raymond. ハッカーズ大辞典. ASCII, 1995.

[7] 青柳隆宏. はじめての OS コードリーディング UNIX V6 で学ぶカーネルのしくみ. 技術評論社, 2013.

[8] Multics. web.mit.edu/multics-history/.

[9] Andrew S. Tanenbaum. OS の基礎と応用 設計から実装、DOS から分散 OS Amoeba まで. 1995.

[10] 藤田昭人. Unix 考古学 Truth of the Legend. ASCII DWANGO, 2016.

[11] Don Libes & Sandy Ressler. *Life with UNIX*. アスキー出版局, 1990.

[12] Brian Kernighan & Dennis Ritchie. プログラミング言語 C 第 2 版 ANSI 規格基準. 共立出版株式会社, 1989.

[13] Peter H. Salus. UNIX の 1/4 世紀. 株式会社アスキー, 2000.

[14] Mike Gancarz. UNIX という考え方. オーム社 開発局, 2001.

[15] 総務省. 平成 27 年度 情報通信白書 通信自由化の背景と 1985 年改革の概要. www.soumu.go.jp/johotsusintokei/whitepaper/ja/h27/html/nc111110.html.

[16] INCORPORATED WESTERN ELECTRIC COMPANY. Software agreement. www.bell-labs.com/usr/dmr/www/licenses/6thEdlicense.pdf, 1974.

[17] John Lions. *Lions' Commentary on UNIX*. アスキー出版局, 1998.

[18] Andrew S. Tanenbaum. オペレーティングシステム 第 2 版 設計と理論および MINIX による実装. ピアソン・エデュケーション, 1998.

[19] Özalp Babaoğlu & William Joy. Converting a swap-based system to do paging in an architecture lacking page-referenced bits. Technical report, Cornell University, 1981.

[20] Marshall Kirk McKusick, Keith Bostic, Michael J. Karels & John S. Quarterman. 4.4BSD の設計と実装. ASCII, 2003.

[21] 総務省. 昭和 57 年版 通信白書 司法省・AT&T 反トラスト訴訟の和解. www.soumu.go.jp/johotsusintokei/whitepaper/ja/s57/html/s57a01010302.html.

[22] Dennis Ritchie. Old licenses and prices. www.bell-labs.com/usr/dmr/www/licenses.html.

[23] Andrew L. Russell. OSI: The Internet That Wasn't. spectrum.ieee.org/tech-history/cyberspace/osi-the-internet-that-wasnt, 2013.

[24] Marshall K. McKusick. バークレー版 UNIX の 20 年. オライリー・ジャパン, 1999.

[25] William Joy. Comments on the performance of UNIX on the VAX. we-

b.archive.org/web/20050306211143/http://www.boulderlabs.com/BILLJOY/joy.pdf.
[26] Richard Stallman. GNU システムとフリーソフトウェア運動. オライリー・ジャパン, 1999.
[27] Free Software Foundation. 自由ソフトウェアとは? www.gnu.org/philosophy/free-sw.ja.html.
[28] Free Software Foundation. コピーレフトって何? www.gnu.org/licenses/copyleft.ja.html.
[29] Free Software Foundation. GNU 一般公衆ライセンス v1.0. www.gnu.org/licenses/old-licenses/gpl-1.0.html.
[30] The NetBSD Project. NetBSD プロジェクトの歴史. www.jp.netbsd.org/ja/about/history.html.
[31] The FreeBSD Project. FreeBSD プロジェクトについて. https://www.freebsd.org/doc/ja_JP.eucJP/books/handbook/history.html.
[32] Chris Demetriou. Announcing the release of NetBSD 1.0. ftp.netbsd.org/pub/NetBSD/misc/release/NetBSD/NetBSD-1.0, 1994.
[33] Eris S. Raymond. 伽藍とバザール. cruel.org/freeware/cathedral.html, 2000.
[34] Alan Hicks, Chris Lumens, David Cantrell & Logan Johnson. Slackware Linux Essentials. www.slackbook.org/html/book.html.
[35] Debian 小史 第 3 章 - Debian リリース. www.debian.org/doc/manuals/project-history/ch-releases.ja.html.
[36] 国領二郎，佐々木裕一，北山聡. Linux はいかにしてビジネスになったか–コミュニティアライアンス戦略. NTT 出版株式会社, 2000.
[37] Eric S. Raymond. 真のプログラマたちの回帰. オライリー・ジャパン, 1999.
[38] Bruce Perens. 「オープンソース」の定義について. オライリー・ジャパン, 1999.
[39] Debian Project. Debian 社会契約. www.debian.org/social_contract.ja.html.
[40] Open Source initiative. The Open Source Definition (Annotated). opensource.org/osd.html.
[41] The NetBSD Project. how netbsd boots on x86. wiki.netbsd.org/how_netbsd_boots_on_x86/.
[42] The NetBSD Project. Chapter 3. Example installation. www.netbsd.org/docs/guide/en/chap-exinst.html.
[43] Luke Mewburn & Matthew Green. build.sh: Cross-building NetBSD. *Proceeding of BSDCon '03*, 2003.
[44] Jordan Hubbard, John Kohl & Hubert Feyrer. Manual page pkg_create(1), 2014.
[45] Manual page deb(5), 2017.
[46] Manual page deb-preinst(5), 2017.
[47] Manual page deb-postinst(5), 2017.
[48] Manual page deb-prerm(5), 2017.
[49] Manual page deb-postrm(5), 2017.
[50] Manual page deb-conffiles(5), 2017.
[51] Free Software Foundation. GNU ライセンスに関してよく聞かれる質問. www.gnu.org/licenses/gpl-faq.ja.html.
[52] 末岡洋子. 「busybox」の gpl 違反訴訟で sfc が勝訴、裁判所が製品の販売停止を命じる. mag.osdn.jp/10/08/05/1045202, 2010.
[53] 岩佐朗子. OSS ライセンス違反とその対策. www.ogis-ri.co.jp/otc/hiroba/technical/oss-license-violations/, 2015.

[54] Free Software Foundation. License:BSD 4Clause. directory.fsf.org/wiki/License:BSD_4Clause.

[55] Open Source Group Japan. licenses/new_BSD_license. ja.osdn.net/projects/opensource/wiki/licenses%2Fnew_-BSD_license.

[56] The FreeBSD Project. PkgBase. wiki.freebsd.org/PkgBase, 2017.

[57] The NetBSD Project. syspkgs. wiki.netbsd.org/projects/project/syspkgs, 2015.

[58] lloyd@must-have-coffee.gen.nz. Lots of broken syspkgs. gnats.netbsd.org/cgi-bin/query-pr-single.pl?number=46937, 2012.

[59] The NetBSD Project. Platforms Supported by NetBSD. www.netbsd.org/ports/.

[60] Yuuki Enomoto & Ken'ichi Fukamachi. Maintain the NetBSD Base System Using pkg_* Tools. *AsiaBSDCon2017 Proceedings*, pp. 127–132, 2017.

[61] Open Source initiative. The 2-Clause BSD License. opensource.org/licenses/BSD-2-Clause.

[62] The pkgsrc-wip project. www.pkgsrc.org/wip/.

[63] The NetBSD Project. New NetBSD developer application procedure. www.netbsd.org/foundation/policies/application-procedure.html.

[64] Brian W. Kernighan & P. J. Plauger. プログラム書法. 共立出版株式会社, 1982.

[65] 松浦智之, 大野浩之, 當仲寛哲. ソフトウェアの高い互換性と長い持続性を目指す POSIX 中心主義プログラミング. デジタルプラクティス, Vol. 8, No. 4, 2017.

[66] Brian W. Kernighan & P. J. Plauger. ソフトウェア作法, 1981.

[67] Dustin Boswell & Trevor Foucher. リーダブルコードより良いコードを書くためのシンプルで実践的なテクニック. オライリー・ジャパン, 2012.

[68] The Origin of Unix Pipes. http://doc.cat-v.org/unix/pipes/.

[69] Vps（仮想専用サーバー）. https://vps.sakura.ad.jp/.

[70] basepkg.netbsd.fml.org/pub/NetBSD/basepkg/7.1/.