第 27 回 東京エリア Debian 勉強会資料

東京エリア デビアン 勉強会

Debian 勉強会幹事 上川 純一

2007 年 4 月 21 日

1 Introduction

上川 純一

今月の Debian 勉強会へようこそ。これから Debian のあやしい世界に入るという方も、すでにどっぷりとつかっているという方も、月に一回 Debian について語りませんか？

目的として次の二つを考えています。

- メールではよみとれない、もしくはよみとってられないような情報について情報共有する場をつくる
- Debian を利用する際の情報をまとめて、ある程度の塊として整理するための場をつくる

Debian の勉強会ということで究極的には参加者全員が Debian Package をがりがりと作るスーパーハッカーになった姿を妄想しています。

Debian をこれからどうするという能動的な展開への土台としての空間を提供し、情報の共有をしたい、というのが目的です。

目次

1 Introduction 1

2 **事前課題** 3
- 2.1 青木さん . . . 3
- 2.2 Kentaro Waki さん . . . 3
- 2.3 森田尚さん . . . 3
- 2.4 ake さん . . . 3
- 2.5 Hashimoto, Toru さん . . . 4
- 2.6 Noriaki Sato . . . 4
- 2.7 鈴木さん . . . 4
- 2.8 H.Honjo さん . . . 5
- 2.9 小室 文さん . . . 5
- 2.10 北原さん . . . 5
- 2.11 小林儀匡 . . . 5
- 2.12 えとーさん . . . 6
- 2.13 keng nak さん . . . 6
- 2.14 でん@相模原さん . . . 6

3 Debian Weekly News trivia quiz 7
- 3.1 2007 年 04 号 . . . 7

4 **最近の** Debian **関連のミーティング報告** 8
- 4.1 東京エリア Debian 勉強会 25 回目報告 . . . 8
- 4.2 東京エリア Debian 勉強会 26 回目報告 . . . 8

5 darcs **使いかた** 10
- 5.1 基本な darcs . . . 10

6 git-buildpackage **の使いかた** 13
- 6.1 git 超入門 . . . 13
- 6.2 git-buildpackage の流れ . . . 16
- 6.3 アップストリームが git で管理している場合 . . . 20

7 git **の実装技術的解説** 22

8 **プロジェクトトラッカーの勧め** 24
- 8.1 プロジェクトトラッカーとは何か . . . 24
- 8.2 なぜトラッカーを使うのか . . . 24
- 8.3 Debian 4.0(“ Etch ”) で利用できるトラッカー (Tracker) は? . . . 24
- 8.4 この資料のコピーライト . . . 26

9 **今後の計画** 27

2 事前課題

上川 純一

今回の事前課題は「私はバージョン管理システムをこのようにつかっています」もしくは「バージョン管理システムを使わなくてすむようにこのようにしています」というタイトルで 200-800 文字程度の文章を書いてください。というものでした。その課題に対して下記の内容を提出いただきました。

2.1 青木さん

VCS は CVS と SVN を使っていますが、あまり複雑なパッチ管理ではなく単に共同開発の最新版管理レベルです。それとバックアップ的な意味合いで使います。ハンドマージより凝ったことといえば VIMDIF の利用ぐらいがせいぜいです。タグ管理はとくいではないです。DVCS をつかうべきかと思いつつ、勉強不足ですね。

2.2 Kentaro Waki さん

2.2.1 私はバージョン管理システムをこのようにつかっています

特に変わった使いかたはしていません。一応、プログラマなのでソースコードのバージョン管理に「subversion」使ってます。つい最近まで「CVS」使ってました。「subversion」で気に入ったのは、「ファイル名の変更」「ブランチの扱い」などです。また、「windows」ユーザと連係する場合に便利そうに見えます。まだ実際には触ってませんが、「TortoiseSVN」は「windows」ユーザに違和感の少ない UIっぽいので。「subversion」関連で、「Trac」の評判も良いようなので導入を検討しています。

2.3 森田尚さん

2.3.1 私はバージョン管理システムをこのようにつかっています

個人で余暇に書いているプログラムのソースコード管理と、仕事（出版社で書籍の企画・編集をしています）での原稿管理のために、Subversion リポジトリを Apache でホストして使っています。2002-2004 年ごろは主に CVS を使っていました。

自分の仕事では、協働するチームメンバが複数の組織と拠点に分散していることが多いので、いずれ分散型 VCS を導入したいと考えています。

2.3.2 バージョン管理システムを使わなくてすむようにこのようにしています

数世代にわたるバックアップが欲しいだけのときなど、VCS を使うまでもない場合のために、ホームディレクトリその他を pdumpfs のバックアップ対象にしています。

2.4 ake さん

「バージョン管理システムを使わなくてすむようにこのようにしています」何故バージョン管理システムが必要なのか？一つの物に複数でよってたかって改変を加えるからである。であれば、使わなくて済むようにするアプローチと

して

- 複数の改変を同時にしない
- 複数の改変が相互に影響しないようにする

のどちらかを行えば良いのである。前者は改変を一人が行うと言う事になるが現実的ではない後者を実現するためには、設計プロセスが完全でなければならない要するに、ソフトウェア開発であれば、モジュール化が適切に行われているなど、設計の瑕疵を無くせばバージョン管理システムは不要である。と言う事に行き着く。実際の所、そう言った状況で作られるソフトウェアはまず無い。多くの開発現場では、混沌の中から偶然の産物の様に作り出されたりする。この状況が存在する限り、やはりバージョン管理システムは必要とされ続けるのだろう。だが設計プロセスを注意深く行えば、設計プロセスの瑕疵を無くす手法が見つかれば、なんとか実現できるかも知れない。

2.5 Hashimoto, Toru さん

今のソフトウェア開発プロジェクトでは、何とバージョン管理システムを使っていない。複数の人が同じファイルを編集しようとすることも実際にあり、その状況はかなり問題である。プロジェクト全体としてバージョン管理を導入する考えはないようである。そこで、自分のチーム内で独自にソースコードを管理する方式を考えているところである。

使用するツールとしては Subversion を考えている。チームの担当範囲のファイルは決まっているので、バージョン管理対象は明確ではあるが、開発環境がよく言えば特殊、悪く言えば洗練されていないために実際にきれいに適用するのは一筋縄ではいかない。こんな環境でもうまく運用できる方法を考えているところである。

その他、自分だけではあるがパッチ管理として quilt を使ってみたことがある。自分の変更点をパッチ (diff) にするのが自動的にできるのは便利だが、自分の作業中に他人が同じファイルを編集するということが起こると混乱してしまった。こういう用途には向かないようだ。

2.6 Noriaki Sato

私はバージョン管理システムをこのようにつかっていたり、いなかったりします（しました）

一年前まで：

バージョン管理システムを使ったことがありませんでした。当時は、実験データ解析用のコードを書いていましたが、`hoge_20051204.cc` のようなファイル名だけで管理していました。（頻繁に書き直したりするわけじゃないので、それで済んでいた）CVS とか名前は知っていたけど、覚えている余裕がありませんでした。

一年前：

仕事で初めて VSS を使いました。チェックアウトするとロックされてしまうのが不便じゃね？と思いました。その後、VSS に上げる前のソースコードを自分のローカルマシン上で管理するために Subversion を使っていました。「達人プログラマ」で、ソースコード以外の普通のドキュメントもバージョン管理システムで管理しよう、と言う話を読んで、なるほど！と思いましたが、結局、今に至るまで実践はしていません。

今：

今度の現場では VSS を使わせてくれないらしいです。それほどコードを書く必要のないプロジェクトなのですが、台帳で管理するとか言う話です。フリーソフトのインストールも不可なので、svn を入れて使ったりする事も出来ません。（一番つらいのは emacs(meadow) を使えない事なのですが、オフトピですね）

2.7 鈴木さん

バージョン管理は、cvs と subversion を使ったことがあります。cvs は、Web(tlec.linux.or.jp) の更新で利用しています。チェックアウトすればどこでも更新できるので便利です。最近余り更新してませんが。subversion は、ドキュメント作成を会社と家で更新できるような 1 日の区切りでチェックインしていました。使ったり使わなかったりで持

続しないです。理由は、リリースするときに tar にまとめてバックアップしているので頻繁に subversion は使ってないです。

2.8 H.Honjo さん

個人で管理しているサーバに、バージョン管理システムとして Subversion を、バグトラックシステムには trac を利用しています。特に変わった用途として使ってはおらず、通常どおりソースコード管理およびバグトラックとして使用しています。主にクライアントマシンとして Windows を使用している関係から、TortoiseSVN を Subversion のクライアントとして利用しています。

新たにリリースされた Etch への移行を検討しており調査中ですが、trac-ja-resource パッケージがうまく動作してくれず、難航しています。

2.9 小室 文さん

2.9.1 バージョン管理システムを使わなくてすむようにこのようにしています

バージョン管理は使っていません。会社ではバージョン管理を使うような人や案件はないです。プライベートでも管理する物も一緒に管理したい人もいないので、導入していません。

バージョン管理を導入する場合、使う人が対等な位置にいるのが前提な気がします。なので、他の案件の一部分の機能を追加するためにファイルを検証サーバにアップする際は (1) 事後報告するか (2) 案件管理者にファイルを送る事が多いです。

使ってみないと！と思いつつも必要に迫られていないのでよく理解していません。

2.10 北原さん

2.10.1 バージョン管理システムを使わなくてすむようにこのようにしています

回答：

個人的に作成しているプログラムはたいした量ではないので、「使わなくてすむように」というよりは、「使う必要がない」という状態です。

プログラムを構成するファイルは、修正前にファイル名にバージョン番号を付けて、全バージョンそのままの形で保存してあります。（若しくは、ディレクトリにバージョン番号を付けて丸ごとコピー。） 規模が小さいのでこれで十分管理できてしまいます。

2.11 小林儀匡

「私はバージョン管理システムをこのようにつかっています」

内容が日々進化していくファイル (プログラム・ドキュメント・翻訳・図など) は何でもリポジトリに入れてバージョン管理下に置いています。バージョン管理システムのない生活はもう考えられなく、バージョン管理システムがあってこそ効率的な仕事ができると思うようになっています。

これまでは主に Subversion などの中央集権的なバージョン管理システムしか使ってきませんでしたが、最近、Debian のウェブサイトやリリースノートの日本語訳コーディネータとして働くようになってから、分散バージョン管理システムにも興味をもつようになりました。翻訳チーム全員にコミット権を与えるわけにはいかないというのはプロジェクトとしては仕方がないことだと思いますが、他方でコーディネータとしては、あらゆるコントリビュータの仕事はきちんと区別し、分割してコミットしたいのです。しかしそのような作業をコーディネータ一人でやるのは大変なので、分散バージョン管理システムを導入して、本家リポジトリへのコミット権はもっていなくても自分のリポジトリにコミットできる翻訳者やレビューアにはどんどん自分で作業をしてもらったほうが、効率がいいのではないかと思っています。

2.12 えとーさん

2.12.1 私はバージョン管理システムをこのようにつかっています

自前のソースコードの管理や参加しているプロジェクトのソースコードの管理に利用しています。

設定ファイルやその他雑多なものはあまり利用できていないので、今後も勉強しながら便利に使っていきたいと思っています。

2.13 keng nak さん

バージョン管理システムは、仕事で cvs をつかっています。ソースコードから word や excel の資料からメモに至るまですべてまとめて cvs で管理していました。今度配属されたプロジェクトでは Mercurial（OpenSolaris などで採用されている分散 SCM）を使用することになり、今まで使ってきた cvs とは勝手が違うために戸惑っております。今回は分散 SCM が 2 種類紹介されるようですので、この機会に分散 SCM の有効な使い方を学べたらと思っています。よろしくお願いします。

2.14 でん@相模原さん

従来、自作プログラムは環境情報をまとめたメモと一緒にソースファイル一式を tar.bz2 で固めて蓄積してました。

この方法ではソースを管理すると言う点では問題は無いのですがそれ以外への発展が無く、方々の作業で無駄が発生しました。代表的なのが、差分抽出により目的外の変更が入っていない事の確認です。このような事をサポートしてくれるツールとしてバージョン管理ツールの DIFF 機能を利用しています。

またバージョン管理業務を長く行っていると

- 「何故バージョンを更新したのか」とか
- 「どうして、このような作りになっているのか」

といった事が忘れてしまいがちです。このためバグトラッキングシステム (Trac) や自動ドキュメント化システム (Doxygen) 等との連携が今後の課題になっています。

今回のお題にはあがりませんでしたが、私は以下のようなシステムで個人的には作業をしています。

- バグトラッキング、仕様書管理　　→ Trac
- バージョン管理　　→ SubVersion/SVK

3 Debian Weekly News trivia quiz

上川 純一

ところで、Debian Weekly News (DWN) は読んでいますか？Debian 界隈でおきていることについて書いている Debian Weekly News. 毎回読んでいるといろいろと分かって来ますが、一人で読んでいても、解説が少ないので、意味がわからないところもあるかも知れません。みんなで DWN を読んでみましょう。

漫然と読むだけではおもしろくないので、DWN の記事から出題した以下の質問にこたえてみてください。後で内容は解説します。

3.1 2007 年 04 号

http://www.debian.org/News/weekly/2007/04/ にある 3 月 13 日版です。

問題 1. ウェブアプリケーション関連のパッケージの静的コンテンツはどこにおくべきか？

□ A /var/www に置く

□ B /usr/share/PACKAGE に置く

□ C /srv/XXX に置く

問題 2. Debian Project の MIA アカウントに対して、実施する WaT とは何をするものか

□ A 今年の DPL 選挙に投票しなかった人に対して確認メールを送り反応がない人を引退プロセスに移行する

□ B 気に入らない人を強制退会させる

□ C あれ？Debian Developer だらけの水泳大会

問題 3. etch リリースはどういう暗号鍵で署名されるか？

□ A オンライン鍵とオフライン鍵

□ B オフライン鍵のみ

□ C オンライン鍵のみ

問題 4. Frans Pop がアナウンスした Babelbox は何をするものか？

□ A いろいろな言葉を喋ってくれる

□ B フォントを複数表示

□ C 自動でくりかえし Debian Installer が稼働し、Gnome にしばらくログインしてくれるしくみ

問題 5. DPL 選挙の勝者は？

□ A Iwamatsu

□ B Sam Hocevar

□ C Anthony Towns

4 最近の Debian 関連のミーティング報告

上川 純一

4.1 東京エリア Debian 勉強会 25 回目報告

東京エリア Debian 勉強会報告。2 月の第 25 回 Debian 勉強会を実施しました。今回は初の小林さんが幹事の会の予定でしたが、小林さんがたおれてしまったので、代理開催です。

今回の参加人数は 13 人でした。あけどさん、小室さん、岩松さん、えとーさん、上川、吉田さん@板橋、Henrich さん、前田さん、石原さん、David Smith さん、澤田さん、キタハラさん、吉田さん（女性）でした。

上川が最近の事情の紹介、事前課題の紹介をしました。「apt に足りない機能」という話題では、非常に盛り上がりました。インストールする前に changelog や README や manpage を表示するためのインタフェースや、google と連携してパッケージをインストールできるようにするインタフェースなどがあるといいね、という話題が出ました。また、ユーザのホームディレクトリにインストールしたパッケージもシステム全体の観点から管理できるとよいねという話題も出ました。

DWN クイズはひさしぶりに DWN が頻繁にリリースされたので、１１問ありました。みなさまの Debian についての常識を問いました。よい感じですね。

dbs について岩松さんが紹介しました。dpatch, quilt によって置き換えられつつある dbs ですが、まだ使っているところもあるので抑えておく必要があります。癖のあるツールですが、この話を聞いてもうみなさん大丈夫ですよね。

そして、上川が dpatch について話をしました。ツールがどういう使い方になるのか、ということと、一つ dbs 風にも使えるのだ、という事例を紹介しました。

最後に、OSC での出し物に付いて議論しました。仮想化については、みなさんすでに活用しまくっているようで、おもしろい話がきけました。Debian ユーザじゃない人たちもくるだろうけど、そういう場合には Windows から Debian に安心して乗り換えてもらえるように goodbye-microsoft.com を紹介しましょう、という話をしました。仮想化の使い道としておもしろいものとして、年賀状、EDY チャージ、winny、試験用（教育）などの事例が出てきました。

4.2 東京エリア Debian 勉強会 26 回目報告

オープンソースカンファレンス (OSC) への仮想化友の会と、東京エリア Debian 勉強会参加報告。3 月の第 26 回 Debian 勉強会を実施しました。今回は仮想化友の会と共催で OSC の会場で開催します。

今回の参加人数は 80 人程度でした。

セミナー会場でのオープニングは仮想化友の会の紹介を平さん、Debian 勉強会の紹介を上川がしました。これでみんな勉強会に参加できるようになったと思います。よろしくおねがいします。

事前課題の声を紹介しました。今回の課題は「仮想化を実際にこういう利用方法で活用しています」でした。さまざまなおもしろい実用例を紹介しました。会場でもそういう使いかたしている、という声が聞こえてきました。別のアーキテクチャのエミュレータを活用して開発だとかカーネルの開発に利用しているという声はあまり聞こえずその利用方法はまだマイナーなようです。

今回は Debian weekly news クイズではなく、「仮想化常識クイズ」を実施しました。全員起立願い、クイズの回答を「グー」「チョキ」「パー」で答えてもらい、正解した人だけのこるという形式でやりました。でんさんが最後まで正解しました。おめでとうございます。

山根さんが「Windows から見える仮想化世界」の紹介をしてくれました。仮想化技術の現状を紹介、でも Windows を使うよりも goodbye-microsoft.com を利用して Debian をインストールしたほうがよいよ、という紹介でした。Debian 勉強会の趣旨に沿った素晴らしい発表です。

今回のメイントピックとして、前田さんに KVM との出会いと活用について語っていただきました。各種 OS を稼働したりした例を紹介しました。

その後はどんどんマニアックな内容に突入します。平さんが「私はこれで○○を辞めました」という発表をしました。素敵な内容でした。

KVM の利用例の紹介として、上川が「KVM で goodbye-microsoft.com を試す」「PaSoRi を試す」実演をしました。

平さんが最後に KVM のソースを読んで解析してみるネタを披露しました。これでもうみんな KVM の起動部分についてハックできるようになったはず。

仮想化友の会と Debian 勉強会はブースもかまえていました。ミニセミナーで、山根さんが自分でパッケージングしたソフトウェアの利用方法を紹介するべく、「2ch リーダー JD の使いかた実演」をしました。

上川が「realcsh, realksh」の紹介をしました。プレゼンテーションは、realksh で 「panic()」するところで終了しました。

懇親会はイタリアンカフェで。あと、一部でベトナム料理を食べにいきました。ふー、長い一日でした。

5 darcs 使いかた

David Smith

darcs はソースコード管理ツールの一つです。Haskell で開発されている注目プロジェクトとして Emacs、Common Lisp、そして Haskell のコミュニティでは大好評です。[*1]

git、Mercurial、monotone、他の数多くの分散ソース管理ツールの仲間であります。しかし、git などの分散ソース管理ツールと違い、バージョンは管理しなく、パッチを管理します。その二つの理念の違い、つまりバージョン管理対パッチ管理、について darcs を活用しながら説明します。

5.1 基本な darcs

5.1.1 とりあえず使ってみよう

先ず、新規リポジトリを作ってみます。

出力

```
exponent,1102:~/workspace> mkdir myproject
exponent,1103:~/workspace> cd myproject
exponent,1104:~/workspace/myproject> darcs ini
exponent,1105:~/workspace/myproject> ls
_darcs
exponent,1106:~/workspace/myproject> echo 'Hello Darcs!' > README
exponent,1107:~/workspace/myproject> darcs whatsnew
No changes!
exponent,1108:~/workspace/myproject> darcs add README
exponent,1109:~/workspace/myproject> darcs whatsnew
\{
addfile ./README
hunk ./README 1
+Hello Darcs!
\}
exponent,1110:~/workspace/myproject> darcs record
Darcs needs to know what name (conventionally an email address) to use as the
patch author, e.g. 'Fred Bloggs <fred@bloggs.invalid>'.  If you provide one
now it will be stored in the file '_darcs/prefs/author' and used as a default
in the future.  To change your preferred author address, simply delete or edit
this file.\footnote{残念ながら、国際化はまだまだ出来ていません。}

What is your email address? David Smith <davidsmith@acm.org>
addfile ./README
Shall I record this change? (1/?)  [ynWsfqadjkc], or ? for help: y

hunk ./README 1
+Hello Darcs!
Shall I record this change? (2/?)  [ynWsfqadjkc], or ? for help: y

What is the patch name? Say hello to darcs
Do you want to add a long comment? [yn]n

Finished recording patch 'Say hello to darcs'
exponent,1111:~/workspace/myproject> darcs changes
Fri Apr 20 15:43:45 JST 2007  David Smith <davidsmith@acm.org>
  * Say hello to darcs
```

darcs ではリポジトリの状態はパッチ集合だけで成り立ちます。パッチを作るのはファイルを darcs に登録し、内容を記録し、最後に名前付けで完成します。ちなみにパッチは名前だけで識別され、バージョン番号は一切ありません。[*2]

[*1] 大好評は多分言い過ぎます。せいぜい部分的に気に入られています。

[*2] 自分の名前とメール先を$HOME/.darcs/author に書いたら聞かれません。

コマンド名	例	意味
init	darcs init	リポジトリを初期化する
whatsnew	darcs whatsnew -v	現在のリポジトリと記録済みの情報との差分を明示する
record	darcs record -m ’バグ#103 を修正’	パッチをリポジトリに記録する
changes	darcs changes –summary	パッチによるチェンジログを生成する
add	darcs add realcsh.c	新しいファイルを darcs に教える
mv	darcs mv realcsh.c dangershell.c	ファイル名前変更

5.1.2 他のリポジトリとの同期

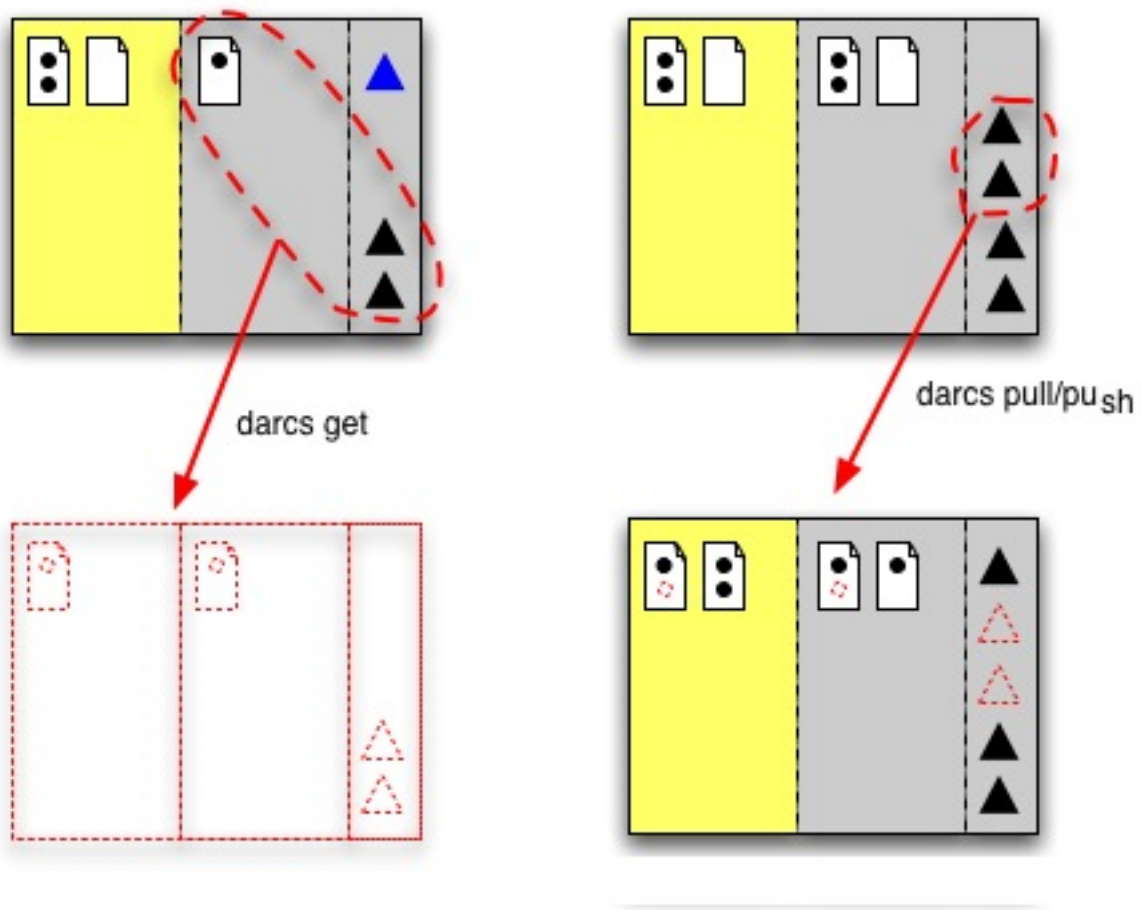
darcs はリポジトリ内のブランチをサポートしてません。複数リポジトリに同じパッチを一個も共有すれば、ブランチと呼んでもいいとのことです。つまり、リポジトリのコピーしかブランチが作れません。darcs get で既に存在するロカールリポジトリ又はネットワークを通して取得出来るリポジトリでコピーできます。[*3]もちろん、ローカルの場合に get を使わず cp でも大丈夫です。

出力

```
exponent,1049:~/workspace> darcs get myproject myproject2
Copying patch 1 of 1... done!
Finished getting.
exponent,1050:~/workspace> cd myproject2
exponent,1051:~/workspace/myproject2> echo ’Darcs rules!’ >> README
exponent,1052:~/workspace/myproject2> darcs record -m ’Add more darcs love’
hunk ./README 2
+Darcs rules!
Shall I record this change? (1/?)  [ynWsfqadjkc], or ? for help: y

Finished recording patch ’Add more darcs love’
exponent,1054:~/workspace/myproject2> darcs push ../myproject

Fri Apr 20 18:04:37 JST 2007  David Smith <davidsmith@acm.org>
  * Add more darcs love
Shall I push this patch? (1/1)  [ynWvpxqadjk], or ? for help: y

Finished applying...

exponent,1055:~/workspace/myproject2>
```

この時、先方のリポジトリに書き込み権利がありましたが、公開されている一般的なプロジェクトでは開発者以外はコミット出来ませんでしょう。そのため、push/pull の代わりに send/apply を使います。send は指定するパッチをメールで送り、apply はファイルにある darcs 形式パッチをリポジトリに記録。

[*3] 現在、HTTP 及び SSH のみの通信になっています。



ここまで darcs の機能がほとんど普通だと言えますでしょう。しかしながらパッチとパッチの依存関係を計算するための「パッチ代数」により Cherry Picking（桜取り）というワークフローが他より余程手軽にできます。残念ですが、例を造るのに darcs が無限ループに落ちているばかりのようなので、とにかく素晴らしいと信用してください。

5.1.3 darcs-buildpackage

Debian パッケージ支持と Haskell を習うために John Goerzen さんが darcs-buildpackage を開発しました。現在、ほとんどソース管理ツールでは Debian パッケージ作業のための'-buildpackage' パッケージはあるよう、その中で darcs-buildpackage の使い方も同様ですので共通できます。

残念ながら、darcs は実際にリポジトリサイズに対して操作がとても遅くなることもあるし、小さいリポジトリでもパッチ代数問題の解決でものすごく負荷かかります。darcs の上流開発は git と Mercurial ほど遅いとも言えますが、理論的にリポジトリの内容に一切障害を与えなく最適化がちゃくちゃく行われています。私としては改善するはずだと思いますが Mercurial に大抵負けています。[*4]

[*4] John Goerzen さんも既に darcs より hg に変更しました

6 git-buildpackage の使いかた

上川 純一

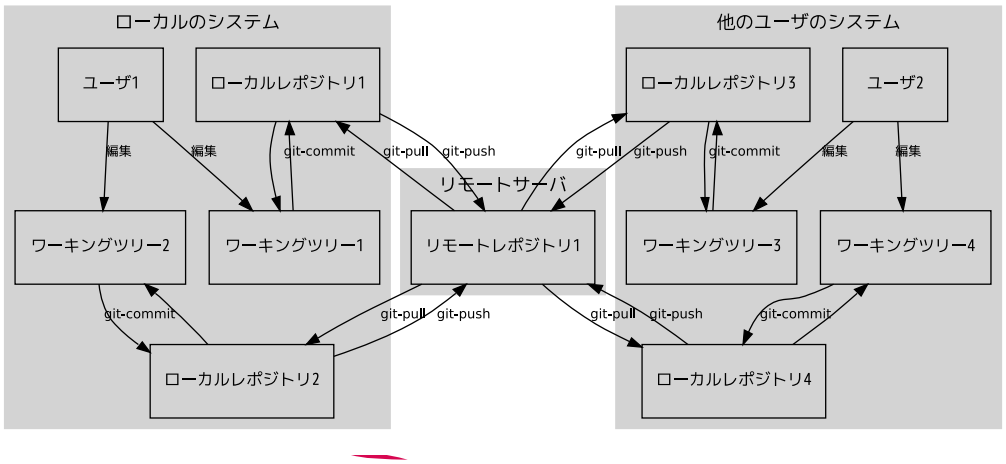
git はソースコードを管理するためのツールです。ソースコード管理のツール、もしくはバージョン管理ツールと呼ばれ、VCS や SCM などと略称されるツールのひとつです。また、旧来の CVS、SVN などの集中モデルの SCM と違い分散モデルを採用しているため、分散 SCM(DSCM) と呼ばれます。 その git を利用して Debian のパッケージを管理するための仕組が git-buildpackage です。

git-buildpackage 利用の詳細に入る前に、基本的な git の利用方法を紹介します。

6.1 git 超入門

6.1.1 git でのデータの流れ

まず最初に git で管理している場合のデータの流れをみてみましょう。各利用者が直接編集しているデータが git-commit や git-push / git-pull コマンドでやりとりされます。[*5]



6.1.2 git 関連用語

ここで、この文書で利用する関連用語を整理しておきます。

[*5] 本当はローカルレポジトリとリモートレポジトリの区別はないため、技術的には直接ユーザ間での git-pull/git-push が可能です。

用語	定義
ワーキングツリー	SCM で管理されている作業用のディレクトリで、ユーザが直接作業できるようにファイルがある場所
ローカルレポジトリ	SCM で管理されているデータワーキングツリーと同じ場所の.git ディレクトリに実体がある。直接ファイルを編集することはできない
リモートレポジトリ	SCM 管理されているデータで、ネットワーク上のどこかに存在しているもの。しばしば他人と共有している。直接ファイルを編集することはできない[*6]
コミット	ワーキングツリーからローカルレポジトリに情報を反映すること
プッシュ	ローカルレポジトリからリモートレポジトリに情報を反映すること
プル	リモートレポジトリからローカルレポジトリとワーキングツリーに情報を反映すること

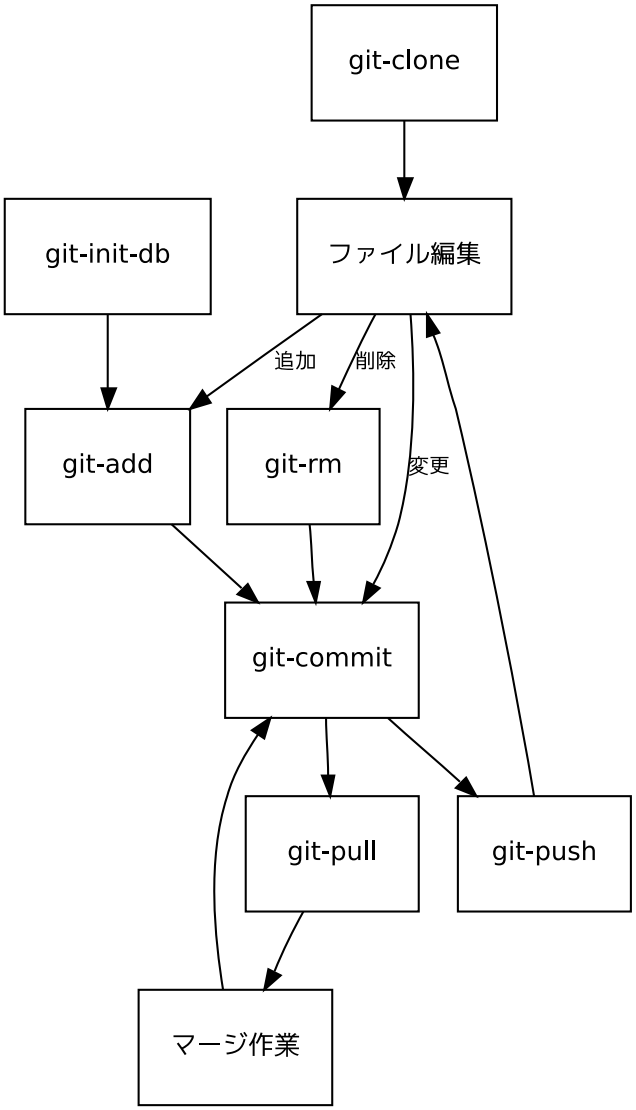
6.1.3 git でよく使うコマンド

git[*7] の基本的な操作はコマンドラインプログラムで実行できるようになっています。git パッケージは多数のコマンドラインのプログラムで構成されています。多数のコマンドはありますが、実は毎日のオペレーションに必要なもの、特に既存の別の SCM から移行してきた場合にいままで行ってきたことと同じことをするために必要なものというのはそれほど多くありません。git でよく使うコマンドを解説します。

コマンド名	例	意味	cvs 相当
git-clone	git-clone *git://XXX/YYY*	リモートレポジトリをローカルにクローンし、ワーキングツリーをチェックアウトする	cvs login, cvs co
git-init-db	git-init-db	ローカルレポジトリ (.git ディレクトリ) を作成する	cvs init
git-pull	git-pull *git://XXX/YYY*	リモートレポジトリの変更をローカルにマージし、ワーキングツリーをアップデートする	cvs up
git-commit	git-commit -a -m 'xxx'	ローカルレポジトリに変更をコミットする	cvs ci の前半
git-push	git-push *git://XXX/YYY*	ローカルレポジトリをリモートレポジトリに送信する	cvs ci の後半
git-add	git-add *filename*	ファイルを次回コミットの際にローカルレポジトリに追加されるように登録する	cvs add
git-rm	git-rm *filename*	ファイルを次回コミットの際にローカルレポジトリから削除されるように登録する	cvs remove
git-status	git-status	ローカルレポジトリに対してワーキングツリーの状況を確認する	cvs status
git-diff	git-diff	ローカルレポジトリとワーキングツリーの差分を表示する	cvs diff

[*7] Debian では `apt-get install git-core` でインストールできる

6.1.4 よくある作業の流れ

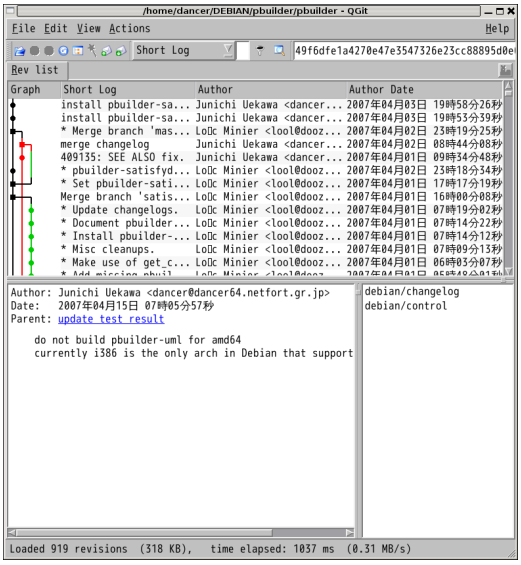


よくある作業の例を紹介します。
まず、新しくローカルレポジトリをつくるのであれば、git-init-db でローカルレポジトリを作成し、ファイルを git-add, git-commit で追加します。そうでなければ、既存のリモートレポジトリを git-clone で複製します。これで、作業可能なワーキングツリーができ、 .git ディレクトリにはローカルレポジトリが作成されました。
ワーキングツリーでファイルを編集して、git-commit でローカルレポジトリに反映します。削除・追加がある場合には、git-add/git-rm コマンドを利用します。通常は、何がコミットされるのか、を git-status コマンドで確認し、ファイルを指定してコミットすることになるでしょう。git-diff コマンドでこれからコミットする予定の差分を確認することもできます。
リモートレポジトリと同期するため、まず git-pull で最新の情報を取得します。ここでコンフリクトがあればワーキングツリーを修正し、git-commit します。問題ないようであれば、git-push でリモートレポジトリに送信します。

6.1.5 GUI ツール

git の履歴情報などを見るのには、視覚的にわかりやすい Qt の GUI である qgit[a] を利用すると履歴や差分が見れて便利です。GUI が利用できない環境では、git-whatchanged コマンドなどを利用すればよいでしょう。他の GUI として git-gui [b]や、gitweb [c]があります。

コミットに特化したツールとして、gct[d] などもあります。



[a] apt-get install qgit でインストール可能

[b] 旧 gitk が git-gui にリネームされた, apt-get install git-gui でインストール可能

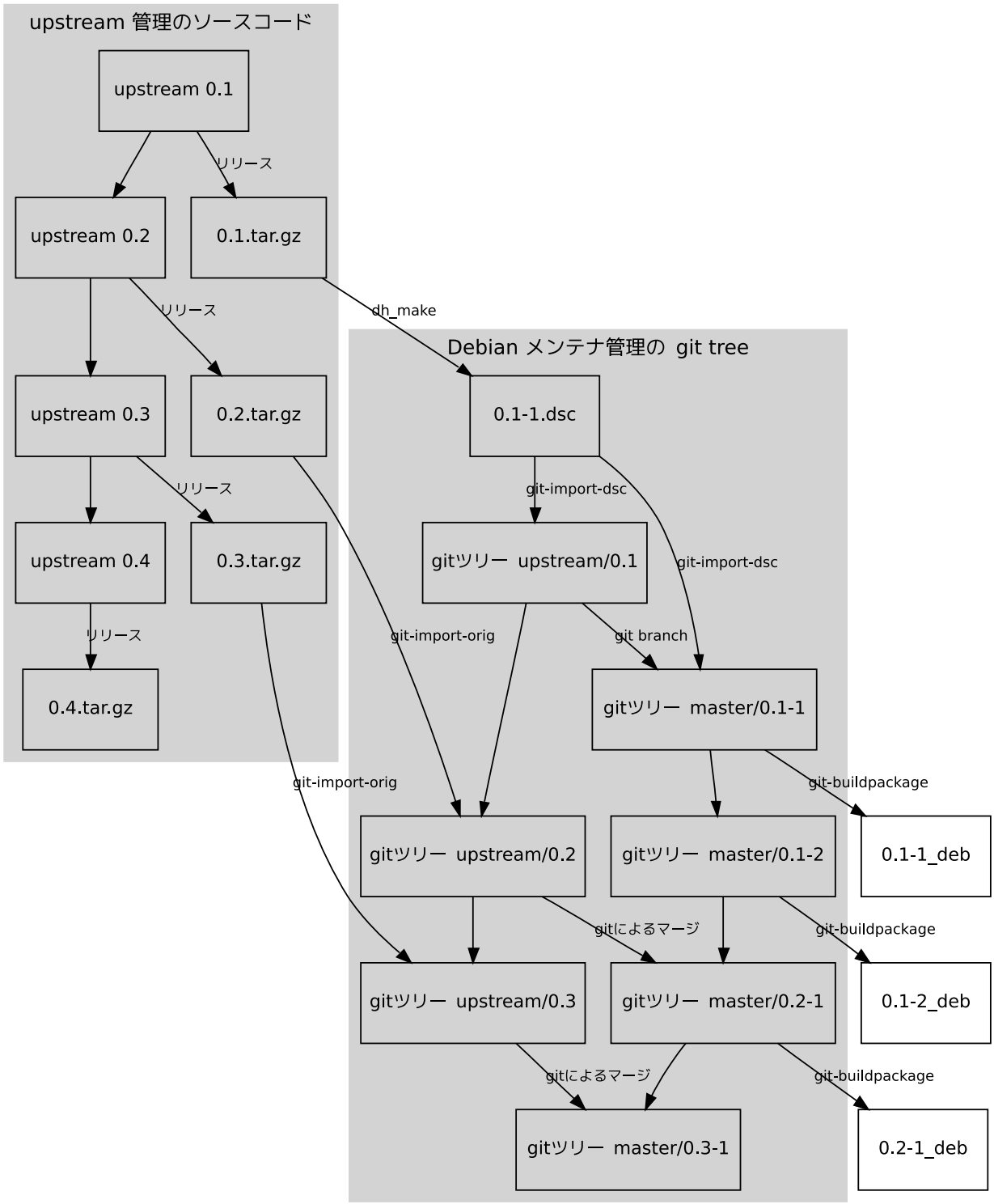
[c] ウェブフロントエンド apt-get install gitweb でインストール。

[d] apt-get install gct

6.2 git-buildpackage の流れ

git-buildpackage[*8] を利用した場合のパッケージングの流れを紹介します。前提として、upstream では謎の利用不可能な SCM を利用して開発をすすめており、Debian Developer はリリース毎に出てくる tar-ball しか利用できないものとします。

[*8] apt-get install git-buildpackage でインストール可能



6.2.1 git-import-dsc

まず、Debian Developer はアップストリームの tarball を展開し、そこで dh_make を実行し、一連のパッケージング作業を行います。

出力

```
[14:10:52]dancer64:work> cp ../upstream/vvv-0.1.tar.gz .
[14:10:57]dancer64:work>
[14:11:03]dancer64:work> tar xfz vvv-0.1.tar.gz
[14:11:08]dancer64:work> cd vvv-0.1
[14:11:12]dancer64:vvv-0.1> dh_make -f ../vvv-0.1.tar.gz

Type of package: single binary, multiple binary, library, kernel module or cdbs?
 [s/m/l/k/b] s

Maintainer name : Junichi Uekawa
Email-Address   : dancer@debian.org
Date            : Sat, 14 Apr 2007 14:11:28 +0900
Package Name    : vvv
Version         : 0.1
License         : blank
Type of Package : Single
Hit <enter> to confirm:
Done. Please edit the files in the debian/ subdirectory now. You should also
check that the vvv Makefiles install into $DESTDIR and not in / .
[14:14:44]dancer64:vvv-0.1> debuild -us -uc
 fakeroot debian/rules clean
dh_testdir
dpkg-buildpackage (debuild emulation): full upload (original source is included)
Now running lintian...
W: vvv: binary-without-manpage usr/bin/hello-world.sh
W: vvv: script-with-language-extension usr/bin/hello-world.sh
E: vvv: helper-templates-in-copyright
E: vvv: description-is-dh_make-template
W: vvv: wrong-bug-number-in-closes l3:#nnnn
E: vvv: section-is-dh_make-template
Finished running lintian.
[14:16:51]dancer64:vvv-0.1> cd ..

[14:17:42]dancer64:work> ls
vvv-0.1                vvv_0.1-1.diff.gz  vvv_0.1-1_amd64.build    vvv_0.1-1_amd64.deb
vvv-0.1.tar.gz        vvv_0.1-1.dsc           vvv_0.1-1_amd64.changes  vvv_0.1.orig.tar.gz
```

すると、Debian のソースパッケージファイルなどができます。

git-import-dsc は dsc ファイルをオプションにとると、[パッケージ名] ディレクトリを作成し、その中に git のローカルレポジトリを作成し、upstream と master ブランチ (通常利用するブランチ) を作成し、upstream にアップストリームのツリーを入れ、master に debian への改変を加えた後のツリーを入れます。この状態で、git-buildpackage などでパッケージがビルドできる状態になっています。

出力

```
[14:17:42]dancer64:work> git-import-dsc vvv_0.1-1.dsc
Upstream version: 0.1
Debian version: 1
Initialized empty Git repository in .git/
Created initial commit b7e3ddd5bb1033c2cd9ae90d86ea8b6f55fb34be
 3 files changed, 16 insertions(+), 0 deletions(-)
 create mode 100644 Makefile
 create mode 100644 hello-world.sh
 create mode 100755 orig.sh
dpkg-source: warning: extracting unsigned source package (/home/dancer/tmp/git-test/work/vvv_0.1-1.dsc)
dpkg-source: extracting vvv in /home/dancer/tmp/git-test/work/tmpwJLVOR/unpack/vvv-0.1-1
dpkg-source: unpacking vvv_0.1.orig.tar.gz
dpkg-source: applying /home/dancer/tmp/git-test/work/vvv_0.1-1.diff.gz
 VCSCMD:  git
LOGTEXT Imported vvv-0.1-1
into Git repository

Created commit 041935d50363f69bcb7ff1b5e0df43b79784b6b1
 9 files changed, 149 insertions(+), 2 deletions(-)
 create mode 100644 debian/README.Debian
[中略]
 create mode 100755 debian/rules
[14:17:58]dancer64:work> cd vvv
[14:18:09]dancer64:vvv> git-status
# On branch master
nothing to commit (working directory clean)
[14:18:12]dancer64:vvv> git-branch
* master
  upstream
```

6.2.2 git-import-orig

新しいアップストリームのバージョンがリリースされた場合には、git-import-dsc で作成されたディレクトリの中で git-import-orig コマンドを実行して、ローカルレポジトリにインポートします。

出力

```
[14:18:50]dancer64:vvv> git-import-orig ../../upstream/vvv-0.2.tar.gz -u 0.2
Upstream version is 0.2
Repository has uncommitted changes, commit them first:
# On branch master
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#        build-stamp
#        configure-stamp
#        debian/files
#        debian/vvv/
nothing added to commit but untracked files present (use "git add" to track)

[14:19:04]dancer64:vvv> debclean
Cleaning in directory ./.git/refs/tags
Directory ./.git/refs/tags: contains no debian/changelog, skipping
Cleaning in directory .
dh_testdir
dh_testroot
rm -f build-stamp configure-stamp
# Add here commands to clean up after the build process.
/usr/bin/make clean
make[1]: ディレクトリ ‘/home/dancer/tmp/git-test/work/vvv’ に入ります
make[1]: ‘clean’ に対して行うべき事はありません.
make[1]: ディレクトリ ‘/home/dancer/tmp/git-test/work/vvv’ から出ます
dh_clean
[14:19:28]dancer64:vvv> git-import-orig ../../upstream/vvv-0.2.tar.gz -u 0.2
Upstream version is 0.2
Importing ../../upstream/vvv-0.2.tar.gz to upstream branch...
Switched to branch "upstream"
  master
* upstream
 VCSCMD:  git
LOGTEXT Imported vvv-0.2
into Git repository



Created commit 8e61554f99a28de9b10c23ae0319b8cc4aa07b40
 2 files changed, 4 insertions(+), 1 deletions(-)
 create mode 100644 NEWS
Merging to master
Switched to branch "master"
* master
  upstream
 100% (14/14) done
Auto-merged Makefile
CONFLICT (content): Merge conflict in Makefile
Automatic merge failed; fix conflicts and then commit the result.
git-pull returned 1
Couldn't pull upstream to .
Import of ../../upstream/vvv-0.2.tar.gz failed
[14:19:50]dancer64:vvv> cat Makefile
all:

install:
<<<<<<< HEAD:Makefile
        install -o root -g root -m 755 hello-world.sh $(DESTDIR)/usr/bin/hello-world.sh
=======
        install -o root -g root -m 755 hello-world.sh /usr/local/bin/hello-world.sh
>>>>>>> 8e61554f99a28de9b10c23ae0319b8cc4aa07b40:Makefile

clean:

.PHONY: all install clean

[14:19:58]dancer64:vvv> vi Makefile
[14:20:15]dancer64:vvv> git-commit -a -m '0.2 debian changes'
Created commit cffcdabb818fdacc38f1a21bab744c56480d1a44
```

この操作により、upstream ブランチに新しいバージョンがインポートされ、master ブランチに変更がマージされ、適切なタグが作成されます。

6.2.3 git-buildpackage

各種編集を行い、コミットを完了したら、git-buildpackage を実行し、レポジトリの内容から Debian パッケージを作成します。–git-tag オプションを付けておくとビルドしなおしたら適切なタグを自動でつけてくれるので便利です。また、コミットしわすれていると警告を出してくれるのもよいです。

出力

```
[14:21:26]dancer64:vvv> git-buildpackage
[中略]
You have uncommitted changes in your source tree:
# On branch master
# Changed but not updated:
#   (use "git add <file>..." to update what will be committed)
#
#       modified:   debian/changelog
#
no changes added to commit (use "git add" and/or "git commit -a")

Use --git-ignore-new to ignore.
[14:21:34]dancer64:vvv> git-commit -a -m 'update changelog'
Created commit 98ae75017264f86192d30b894191d862b98821f5
 1 files changed, 6 insertions(+), 0 deletions(-)
[14:21:43]dancer64:vvv> git-buildpackage
dh_testdir
dh_testroot
rm -f build-stamp configure-stamp
[中略]
$ git-buildpackage -us -uc --git-tag
```

6.3 アップストリームが git で管理している場合

別のシナリオを考えてみます。アップストリームが git で管理している場合に直接 git を利用したワークフローは幾分か簡単にできます。どちらがよいかはわかりませんが、参考までに掲載します。

git-clone したらローカルレポジトリはリモートレポジトリから見たらただのブランチのため、ローカルレポジトリで変更を加えていれば、git-pull するたびにアップストリームの変更をマージすることになります。ビルドする際には、debuild などを直接使えばよいでしょう。ただ、この場合は自動でタグをつけたりはしてくれません。

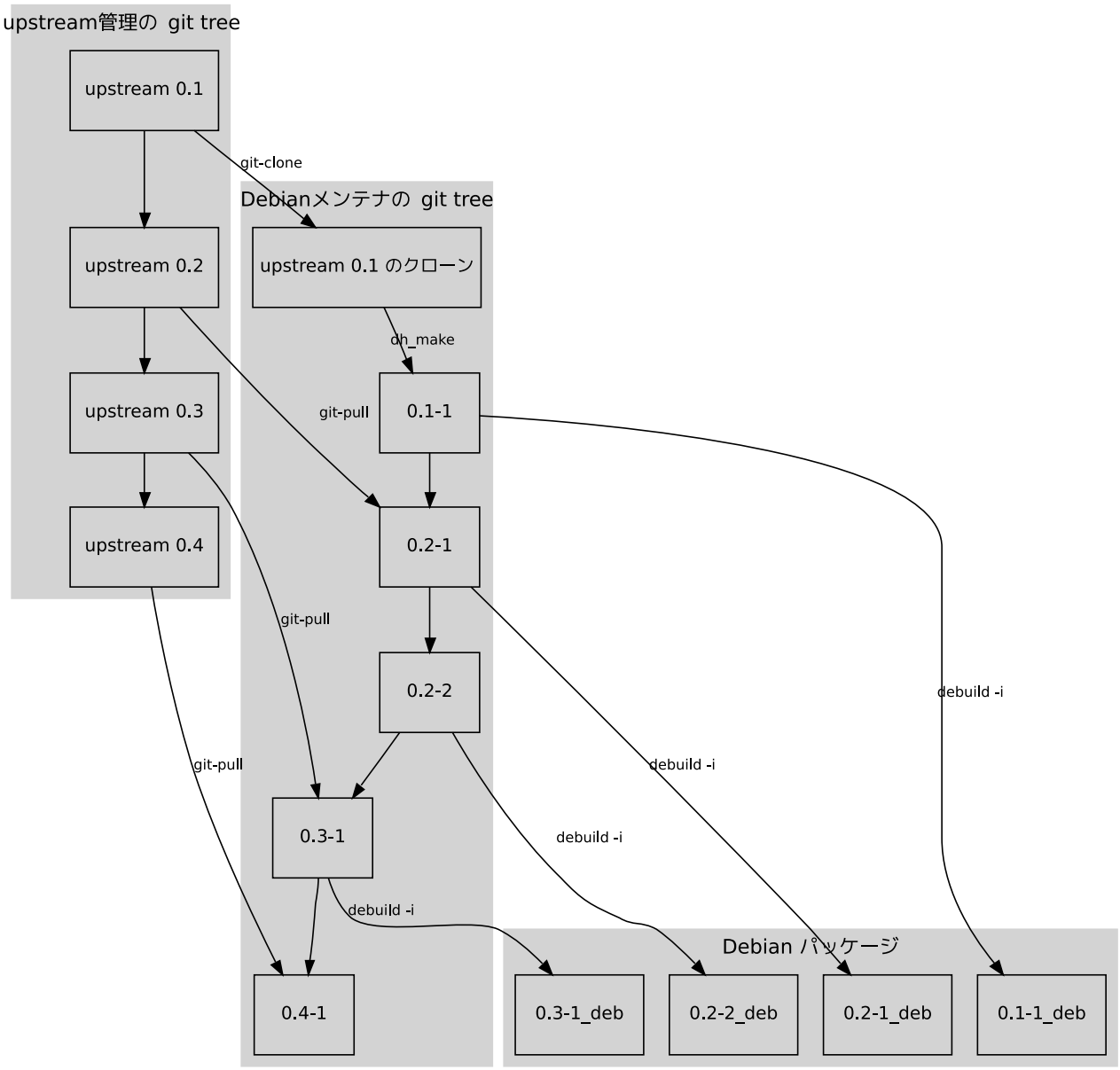
出力

```
$ debuild -us -uc -i -I
```

そのため、git-import-dsc, git-import-orig を使わない場合においても git-buildpackage を利用するのがよいでしょう。upstream ブランチが必要になるのは git-import-orig をする際だけなので、master ブランチのみしかなくても動きます。

出力

```
$ git-buildpackage -us -uc -i -I
```

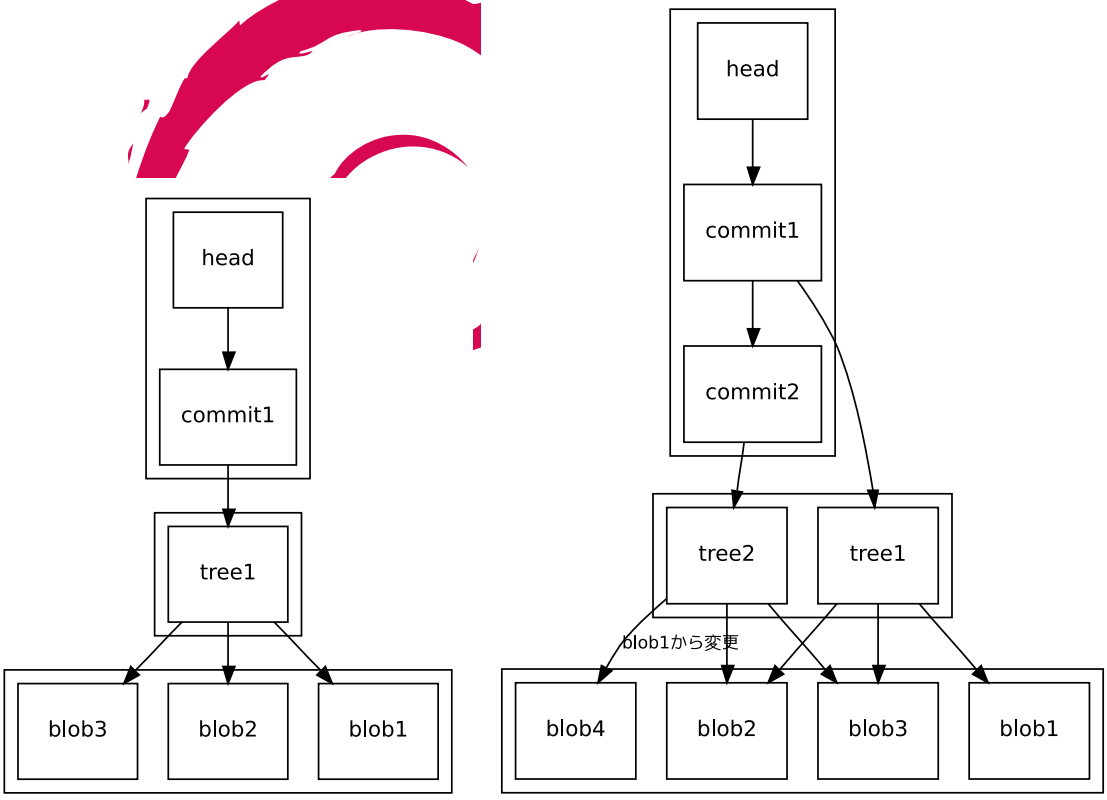

upstream管理の git tree
upstream 0.1
git-clone
Debianメンテナの git tree
upstream 0.2
upstream 0.1 のクローン
dh_make
upstream 0.3
git-pull
0.1-1
upstream 0.4
0.2-1
git-pull
0.2-2
debuild -i
git-pull
debuild -i
0.3-1
debuild -i
debuild -i
Debian パッケージ
0.4-1
0.3-1_deb
0.2-2_deb
0.2-1_deb
0.1-1_deb

7 git の実装技術的解説

上川 純一

簡単に git のレポジトリ[*9]のファイル構成のコンセプトを紹介します。レポジトリのファイルは .git ディレクトリ以下[*10]にあります。基本的なオブジェクトは commit, tree, blob の三種類があります。. git/objects/ 以下にファイルは gzip 圧縮された状態で保存されています。. git/HEAD (最近は.git/refs/heads/master) に最新の commit のハッシュ（ハッシュはデータの sha1sum をとることで取得している）が記録されています。commit の中身を見ると、tree のハッシュと 親 commit とコミットメッセージ情報が含まれています。tree を見ると、そのコミットのときのディレクトリ情報が含まれており、それぞれのファイル実体（blob）の hash 値が含まれています。

なお、objects はばらばらで管理されるとディスクの利用効率が悪いので git-repack コマンドを利用して pack 形式で保存されることが多いです。そうなると、.git/objects/pack に保存されます。



この情報を具体的にコマンドラインで確認してみた例を例示します。特に重要な点は、'head' ファイルへの書き込みという OS 観点ではアトミックに実装できるオペレーションでコミットが実現されているということでしょう。[*11]

[*9] ローカルレポジトリもリモートレポジトリも基本的には同じファイル構成です。
[*10] もしくは環境変数 GIT_DIR で指定された場所
[*11] ファイルの SHA-1 ハッシュ値の衝突が発生しないということが前提です。ただ、現実的に問題になることは無いでしょう。

出力

```
[11:39:25]dancer64:pbuilder> cat .git/refs/heads/master
49f6dfe1a4270e47e3547326e23cc88895d0e05d
[11:39:38]dancer64:pbuilder> git-cat-file -t 49f6dfe1a4270e47e3547326e23cc88895d0e05d
commit
[11:39:44]dancer64:pbuilder> git-cat-file commit 49f6dfe1a4270e47e3547326e23cc88895d0e05d
tree 98dc4f51c0ae895607db43f8b617a7cacfbdc34b
parent e40c851b3017b09a609e2e36af6ae8a20b8ffff3
author Junichi Uekawa <dancer@dancer64.netfort.gr.jp> 1176588357 +0900
committer Junichi Uekawa <dancer@dancer64.netfort.gr.jp> 1176588357 +0900

do not build pbuilder-uml for amd64
currently i386 is the only arch in Debian that supports user-mode-linux
[11:39:47]dancer64:pbuilder> git-cat-file -t 98dc4f51c0ae895607db43f8b617a7cacfbdc34b
tree
[11:39:52]dancer64:pbuilder> git-cat-file tree 98dc4f51c0ae895607db43f8b617a7cacfbdc34b|strings|head -5
100644 .cvsignore
100644 .gitignore
100644 AUTHORS
o[v'a
R100644 COPYING
[11:40:13]dancer64:pbuilder> git-ls-tree 98dc4f51c0ae895607db43f8b617a7cacfbdc34b|head -5
100644 blob 99ee691a57a1d79999965e8f9362da79d18e8ce4    .cvsignore
100644 blob e4e5f6c8b2deb54bf38312dd9e2f53489b60d6a6    .gitignore
100644 blob 30ded29ac4176f5b7627618b27a8cb15c7ea0a52    AUTHORS
100644 blob b7b5f53df1412df1e117607f18385b39004cdaa2    COPYING
100644 blob 97cae37dd0355d4f29837bc02bbad4bcdba98914    ChangeLog
[11:40:18]dancer64:pbuilder> git-cat-file -t 97cae37dd0355d4f29837bc02bbad4bcdba98914
blob
[11:40:36]dancer64:pbuilder> git-cat-file blob 97cae37dd0355d4f29837bc02bbad4bcdba98914|head
2007-04-11  Junichi Uekawa  <dancer@debian.org>

        * AUTHORS, etc: remove $Id$, which is CVS specific

2007-04-10  Junichi Uekawa  <dancer@debian.org>

        * Documentation/pbuilder-doc.xml: update documentation

        * pbuilder-modules: say lenny instead of sarge

[11:40:43]dancer64:pbuilder> ls .git/objects/97/cae37dd0355d4f29837bc02bbad4bcdba98914
.git/objects/97/cae37dd0355d4f29837bc02bbad4bcdba98914
[11:40:58]dancer64:pbuilder> ls -l .git/objects/97/cae37dd0355d4f29837bc02bbad4bcdba98914
-r--r--r-- 1 dancer dancer 27255 2007-04-11 09:01 .git/objects/97/cae37dd0355d4f29837bc02bbad4bcdba98914
```

8 プロジェクトトラッカーの勧め

矢吹 幸治 (yabuki@netfort.gr.jp)

8.1 プロジェクトトラッカーとは何か

プロジェクトトラッカーは、ある作業にかかった時間を記録するためのプログラムです。

8.2 なぜトラッカーを使うのか

このツールは、作業時間を計測するツールです。このトラッカーを使うことにより、

1. 自分の能力を高める。
2. ある作業を開始して完了するまでの時間を予想しやすくする。
3. 自分の時間の使い方を見直すことができる。

という利点があります。欠点としては、自分の作業を記録するのがめんどくさい という点があります。しかし記録する利点に比べれば、かける手間は引き合います。

また、他人に私を観察してもらうよりは、自分で私を観察するほうが、己の自尊心のために安全です。— 「他人に指摘してもらう」ことは多くの人にとって苦痛で、受け入れ難い事です。

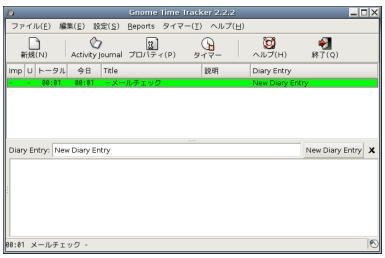
8.3 Debian 4.0(“ Etch ”) で利用できるトラッカー (Tracker) は?

以下の 4 つがある

- gnotime – GTK ベース
- karm – Qt ベース
- wmwork – X ベース
- worklog – CUI ベース

8.3.1 gnotime

GTK で書かれた、GNOME と親和性の高いプロジェクトトラッカー。日本語も使えて、活動履歴 (Activity Journal) も出力できる。Desktop で GNOME を使っているなら、おすすめ。



8.3.2 karm

Qt で書かれたプロジェクトトラッカー。たぶん便利だと思う。今回は時間切れで試さず。私は KDE 使いじゃないのでだれか、小ネタでいいので発表で使い勝手レポートをしてくれるとうれしい。

8.3.3 wmwork

X があれば動く、プロジェクトトラッカー。

インストール方法は、

出力

```
aptitude install wmwork
```

Small is beautiful. シンプルで Windowmaker や blackbox などのシンプルな window manager と相性が良さそう。

設定は、wmwork が起動していないときに、/.wmwork/wmworklog を編集する。詳細は、/usr/share/doc/wmwork/配下のドキュメントを参照せよ。以下は、/usr/share/doc/wmwork/exsample/worklog を引用した。

出力

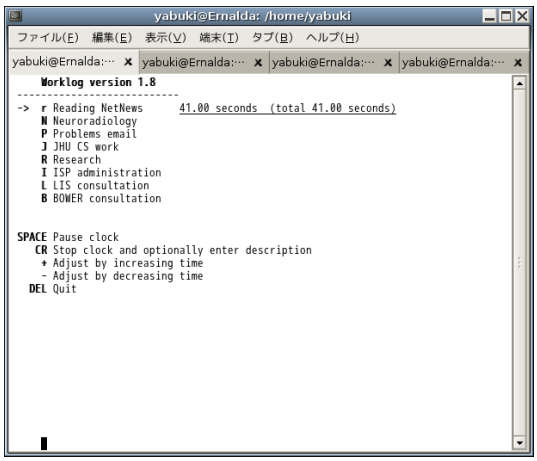
```
1# sample wmwork configuration file
2# do not edit while wmwork is running
3#
4# you may save this file as an initial ~/.wmwork/worklog
5
6A02:0:comment here
7PSI:0
8TEST:0:only 'TES' will be shown
```

上記の設定ファイルをみると判るが、最初の 3 letter が wmwork の表示に使われる。識別子というべきか。また、この最初の部分に指定したファイル名でログが取られる。次に費した時間 (秒単位)、最後にコメントである。

~/.wmwork/以下にロックファイルができて、2 重起動ができないようになっている。

8.3.4 worklog

CUI で動かせる。ssh などでサーバに入って作業をしても、このプログラムなら利用できる。



出力

```
aptitude install worklog
```

このプログラムには、起動するまでに設定ファイルを作成しておく必要がある。詳しくは、/usr/share/doc/worklog/のファイル群を読むこと。設定ファイルの書式は、

出力

```
キー:割り当てる名前
```

です。例えば

出力

```
B:BOWER consultation
L:LIS consultation
R:Research
r:Read NetNews
```

であれば、B,L,R,r のキーで対象となる時間を切替えながら計測できます。残念ながら表示には日本語 (UTF-8) は通りません。(ログには日本語 (UTF-8) がそのまま出ているので lv などでは読めました。

/usr/share/doc/worklog/examples/projects に雛型ファイルがあるので、これをコピーして改変して使うのがよいでしょう。

出力

```
alias wl='worklog $HOME/logs/worklog.projects $HOME/logs/worklog.time.log'
```

がお薦めの alias だそうです。確かに、設定ファイルのキーバインド毎にログが生成されるので、ディレクトリを作っておくのが良いでしょう。

8.4 この資料のコピーライト

この資料は、GPL で配布します。不明点などあれば、メールなどの手段で連絡をください。[*12]

[*12] 矢吹 幸治 (yabuki@netfort.gr.jp)

9 今後の計画

上川 純一

- 5 月：
- 6 月：スコットランドで開催
- 7 月：
- 8 月：
- 8 月：
- 8 月：
- 8 月：
- 12 月：反省会

デビアン勉強会

Debian 勉強会資料

2007 年 4 月 21 日　　初版第 1 刷発行
東京エリア Debian 勉強会 （編集・印刷・発行）