Special Feature 1
→Gitを使いこなしていますか？

Special Feature 2
→Vuls爆誕！

Software Design

2017年10月18日発行
毎月1回18日発行
通巻390号
（発刊324号）
1985年7月1日
第三種郵便物認可
ISSN 0916-6297
定価
本体 1,220円
+税

［ソフトウェア デザイン］
OSとネットワーク、
IT環境を支える
エンジニアの総合誌

October 2017
10

個人でも会社でも使える！

Special Feature 01
これだけは知っておきたい
Gitのキホン
GitHub/GitLab/Bitbucket/GitBucket

Special Feature 02
脆弱性スキャナ Vuls入門
システムのセキュリティチェックをもっと楽に

あなたのサーバは本当に安全ですか？



Extra Feature 01
AWS Lambdaで実現した画像管理環境

Extra Feature 02
ネットワークエンジニア技術の極め方

Extra Feature 03
Telnetでメールサーバと通信してみる



この個所は、雑誌発行時には広告が掲載されていました。編集の都合上、
総集編では収録致しません。

目次



Contents

Software Design

October, 2017

10

Special Feature 1 → 第1特集 | Page

17

これだけは知っておきたい Gitのキホン

個人でも会社でも使える！

GitHub/GitLab/Bitbucket/GitBucket

第1章 GitHub、Bitbucket、GitLab、GitBucket徹底比較! 竹添 直樹 18
Gitサービス／プロダクトの選び方

第2章 Gitの基礎から実践までを学ぼう 豊沢 泰尚、久保 靖資 26
インストールから複数人での共同作業運用までのGit操作

第3章 構造を知るとGitの真価がよくわかる 豊沢 泰尚、久保 靖資 39
Gitの内部構造とコマンドの作用の背景

第4章 強力プラグインgina.vim Alisue 46
実践! エディタからのGit操作[Vim編]

第5章 GitからGitHubまでエディタから操作して作業がはかどる! 大竹 智也 53
実践! エディタからのGit操作[Atom編]

第6章 Git活用の実例[1] 山際 康貴 60
クラウド=GitHubとオンプレ=GitLabを使い分けるピクシブ

第7章 Git活用の実例[2] 打田 智子 64
GitLab CI+RedPenでドキュメント検査もCIしよう!

目次

Contents



Software Design

October, 2017

10

Special Feature 2 → 第2特集 | Page

システムのセキュリティチェックをもっと楽に
脆弱性スキャナVuls入門 69

第1章 セキュリティ対応に追われる現場の悲鳴と、その解決策 林 優二郎 70

第2章 Vuls導入チュートリアル 枇榔 晃裕、牛田 隆之 75

第3章 Vulsとさまざまなプロダクトとの外部連携 牛田 隆之 83

第4章 Vuls誕生秘話——OSSをバズらせるには 神戸 康多 88

第5章 実際の運用と、今後のVulsの進化 福田 鉄平 91

Extra Feature → 一般記事 | Page

サーバーレスで実現! 素材集サービスを効率化した自動画像管理システムに学ぶ Serverless Framework開発入門 星 直史 96

スキルアップの傾向と対策 ネットワークエンジニア その技術の極め方 釜山 公徳 106

telnetを武器にシステムの本質を理解する 身近なメールで使われている技術をネットワークコマンドで体験してみよう 中井 勘介 112

Catch Up Trend | Page

うまくいくチーム開発のツール戦略［10］ CIツール「Bamboo」で開発プロセスの生産性を上げる 河村 康博 180

Test Report | Page

NETGEAR ReadyNAS徹底運用［3］ ReadyNASのデータ復元 中山 一弘 184

à la carte → アラカルト | Page

ITエンジニア必須の最新用語解説［106］railcar 杉山 貴章 ED-1

読者プレゼントのお知らせ 16

バックナンバーのお知らせ 68

SD BOOK REVIEW 111

SD NEWS & PRODUCTS 188

Readers' Voice 190

Column

		Page
digital gadget [226] コンピュータグラフィックスの祭典SIGGRAPH 2017[前編]	安藤 幸央	1
結城浩の再発見の発想法 [53] Checksum——チェックサム	結城 浩	4
及川卓也のプロダクト開発の道しるべ [12] プロダクトマネージャーの実務	及川 卓也	6
宮原徹のオープンソース放浪記 [20] 今年のOSC京都も暑かった!	宮原 徹	10
ツボイのなんでもネットにつなげちまえ道場 [28] BBC micro:bitを使ってみる	坪井 義浩	12
ひみつのLinux通信 [44] りなてえぴっく(Linux Team EPIC)	くつなりょうすけ	95
Hack For Japan〜あなたのスキルは社会に役立つ [70] 盛り上がる技術系同人活動に乗っかれ!「Code for 同人誌」活動報告	小泉 勝志郎	174
温故知新 ITむかしばなし [70] 仮想86モード〜x86は、どのように16bit互換性を維持してきたのか	速水 祐	178

Development

		Page
書いて覚えるSwift入門 [30] SD読者ならわかるブロックチェーン入門	小飼 弾	117
シェル芸人からの挑戦状 [2] 暦や日付の計算	山田 泰宏、eban、中村 壮一、 田代 勝也、上田 隆一	122
RDBアンチパターン [6] ソートの依存	曽根 壮大	128
Androidで広がるエンジニアの愉しみ [19] セーラー服を彩るフルカラーLEDを制御する	鈴峰 きり、 小暮 敦彦、林 博	136
Vimの細道 [22] foldでテキストを折り畳む(前編)	mattn	140
セキュリティ実践の基本定石 [48] ソフトウェアのサプライチェーンとそこに潜む脅威	すずきひろのぶ	144

OS/Network

		Page
Debian Hot Topics [49] Debian 10 "Buster" に向けた取り組みが続々始動	やまねひでき	148
Ubuntu Monthly Report [90] OpenVPNでおうちネットワークにログイン	あわしろいくや	152
SOURCES〜レッドハット系ソフトウェア最新解説 [13] OpenShiftの新機能 Ansible Service Broker	小島 啓史	156
Linuxカーネル観光ガイド [66] RACKによるTCPのパケットロス検出	青田 直大	159
Unixコマンドライン探検隊 [18] ゲーム&ジョーク特集(その3)	中島 雅弘	166
Monthly News from jus [72] 地域を超え、ますます激しさが増すシェル芸勉強会	小林 峻	172



[広告索引]
システムワークス
http://www.systemworks.co.jp/
前付1

創夢
http://www.soum.co.jp/
表紙の裏

日本コンピューティングシステム
http://www.jcsn.co.jp/
裏表紙の裏

ネットギア
https://www.netgear.jp/
裏表紙

[ロゴデザイン]
デザイン集合ゼブラ+坂井 哲也
[表紙デザイン]
藤井 耕志(Re:D Co.)
[表紙写真]
Dixi / Adobe Stock
[イラスト]
*フクモトミホ
*高野 涼香
[本文デザイン]
*安達 恵美子
*石田 昌治(マップス)
*岩井 栄子
*ごぼうデザイン事務所
*近藤 しのぶ
*SeaGrape
*轟木 亜紀子、阿保 裕美、
佐藤 みどり、徳田 久美
(トップスタジオデザイン室)
*伊勢 歩、横山 慎昌(BUCH+)
*藤井 耕志、萩村 美和(Re:D Co.)
*森井 一三



この個所は、雑誌発行時には広告が掲載されていました。編集の都合上、
総集編では収録致しません。

GIHYO Digital Publishing

技術評論社の本が電子版で読める!

https://gihyo.jp/dp

イチオシの1冊!

IBM Bluemix クラウド開発入門 —Webから拡張知能Watsonまで実践解説

常田秀明，水津幸太，大島騎頼　著，Bluemix User Group　監修

2,800円 EPUB PDF

IBMのクラウドサービスであるBluemixを，基本的な導入方法の解説から実際のアプリケーションを作る方法まで本書では紹介します。Bluemixの特徴はさまざまな事例に支えられた豊富なサービス群です。アプリケーションを開発・運用するためのDevOpsについて工夫が凝らされており，最近話題のAI:拡張知能利用のためのWatsonAPI等が提供されてます。IoTについても各種の対応が施されており，本書ではRaspberry PiとWatsonを組み合わせた事例を紹介しています。スマートなクラウド利用の手引きとしてご活用ください。

https://gihyo.jp/dp/ebook/2017/978-4-7741-9126-3



あわせて読みたい



［改訂新版］
C# ポケットリファレンス
EPUB PDF



かんたん UML 入門［改訂2版］
EPUB PDF



現場で役立つシステム設計の原則
~変更を楽で安全にするオブジェクト指向の実践技法
EPUB PDF



きちんとわかる! JavaScript とことん入門
EPUB PDF

他の電子書店でも好評発売中!

amazon kindle　honto　楽天 kobo　ヨドバシカメラ www.yodobashi.com　BookLive!

お問い合わせ

〒162-0846　新宿区市谷左内町21-13　株式会社技術評論社　クロスメディア事業部
TEL：03-3513-6180　メール：gdp@gihyo.co.jp
法人などまとめてのご購入については別途お問い合わせください。

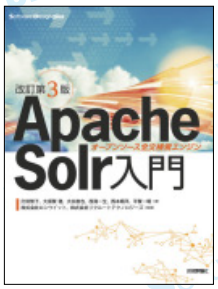
Software Design plus

Software Design plusシリーズは、OSとネットワーク、IT環境を支えるエンジニアの総合誌『Software Design』編集部が自信を持ってお届けする書籍シリーズです。

最新刊！



常田秀明、水津幸太、大島騎頼 著、Bluemix User Group 監修
B5変形判・288ページ
定価 2,800円（本体）+税
ISBN 978-4-7741-9084-6



打田智子、大須賀稔、大杉直也、西潟一生、西本順平、平賀一昭 著
B5変形判・392ページ
定価 3,800円（本体）+税
ISBN 978-4-7741-8930-7



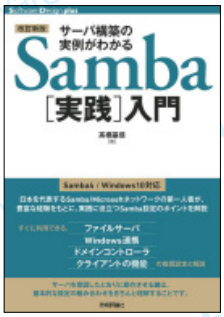
養成読本編集部 編
B5判・144ページ
定価1,780円（本体）+税
ISBN 978-4-7741-8865-2



星野武史 著、花井志生 監修
A5判・256ページ
定価 2,580円（本体）+税
ISBN 978-4-7741-8729-7



小川晃通 著
A5判・272ページ
定価 2,280円（本体）+税
ISBN 978-4-7741-8570-5



髙橋基信 著
A5判・256ページ
定価 2,680円（本体）+税
ISBN 978-4-7741-8000-7



山本小太郎 著
B5変形判・176ページ
定価 2,480円（本体）+税
ISBN 978-4-7741-8885-0



中井悦司 著
B5変形判・272ページ
定価 2,980円（本体）+税
ISBN 978-4-7741-8426-5



前橋和弥 著
B5変形判・304ページ
定価 2,680円（本体）+税
ISBN 978-4-7741-8188-2



五味弘 著
B5変形判・272ページ
定価 2,580円（本体）+税
ISBN 978-4-7741-8035-9



養成読本編集部 編
B5判・168ページ
定価1,980円（本体）+税
ISBN 978-4-7741-8360-2



養成読本編集部 編
B5判・232ページ
定価2,080円（本体）+税
ISBN 978-4-7741-8385-5



養成読本編集部 編
B5判・200ページ
定価2,080円（本体）+税
ISBN 978-4-7741-8034-2



養成読本編集部 編
B5判・112ページ
定価 1,980円（本体）+税
ISBN 978-4-7741-7992-6

Dockerエキスパート養成読本
養成読本編集部 編
定価 1,980円+税　ISBN 978-4-7741-7441-9

サーバ／インフラエンジニア養成読本 基礎スキル編
福田和宏、中村文則、竹本浩、木本裕紀 著
定価 1,980円+税　ISBN 978-4-7741-7345-0

Laravelエキスパート養成読本
川瀬裕久、古川文生、松尾大、竹澤有貴、小山哲志、新原雅司 著
定価 1,980円+税　ISBN 978-4-7741-7313-9

Pythonエンジニア養成読本
鈴木たかのり、清原弘貴、嶋田健志、池内孝啓、関根裕紀、若山史郎 著
定価 1,980円+税　ISBN 978-4-7741-7320-7

データサイエンティスト養成読本 R活用編
養成読本編集部 編
定価 1,980円+税　ISBN 978-4-7741-7057-2

Javaエンジニア養成読本
きしだなおき、のざきひろふみ、吉田真也、菊田洋一、渡辺修司、伊賀敏樹 著
定価 1,980円+税　ISBN 978-4-7741-6931-6

JavaScriptエンジニア養成読本
吾郷協、山田順久、竹馬光太郎、智大二郎 著
定価 1,980円+税　ISBN 978-4-7741-6797-8

WordPress プロフェッショナル養成読本
養成読本編集部 編
定価 1,980円+税　ISBN 978-4-7741-6787-9

サーバ／インフラエンジニア養成読本 ログ収集～可視化編
養成読本編集部 編
定価 1,980円+税　ISBN 978-4-7741-6983-5

フロントエンドエンジニア養成読本
養成読本編集部 編
定価 1,980円+税　ISBN 978-4-7741-6578-3

PHPライブラリ&サンプル実践活用［厳選100］
WINGSプロジェクト 著
定価 2,480円+税　ISBN 978-4-7741-6566-0

Hyper-V仮想化技術活用ガイド
遠山藤乃 著
定価 3,500円+税　ISBN 978-4-7741-6571-4

改訂版　Zabbix統合監視実践入門
寺島広大 著
定価 3,500円+税　ISBN 978-4-7741-6543-1

Vyatta仮想ルータ活用ガイド
松本直人、さくらインターネット研究所、日本Vyattaユーザー会 著
定価 3,300円+税　ISBN 978-4-7741-6553-0

技術評論社



養成読本編集部 編
B5判・112ページ
定価2,180円(本体)+税
ISBN 978-4-7741-8894-2



養成読本編集部 編
B5判・192ページ
定価1,980円(本体)+税
ISBN 978-4-7741-8863-8



養成読本編集部 編
B5判・160ページ
定価2,180円(本体)+税
ISBN 978-4-7741-8895-9



養成読本編集部 編
B5判・240ページ
定価 1,980円(本体)+税
ISBN 978-4-7741-8877-5



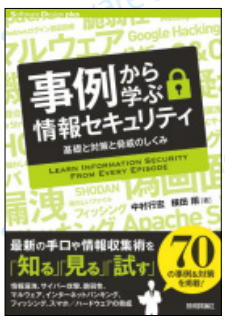
香山哲司、小野寺匠 著
A5判・176ページ
定価 2,480円(本体)+税
ISBN 978-4-7741-8815-7



高宮安仁、鈴木一哉、松井暢之、
村木暢哉、山崎泰宏 著
A5判・352ページ
定価 3,200円(本体)+税
ISBN 978-4-7741-7983-4



斎藤祐一郎 著
A5判・160ページ
定価 2,280円(本体)+税
ISBN 978-4-7741-7865-3



中村行宏、横田翔 著
A5判・320ページ
定価 2,480円(本体)+税
ISBN 978-4-7741-7114-2



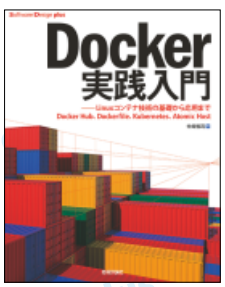
川本安武 著
A5判・400ページ
定価 2,980円(本体)+税
ISBN 978-4-7741-6807-4



勝俣智成、佐伯昌樹、
原田登志 著
A5判・288ページ
定価 3,300円(本体)+税
ISBN 978-4-7741-6709-1



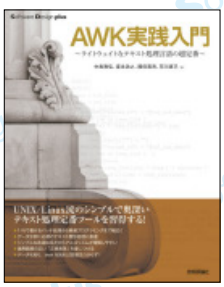
神原健一 著
B5変形判・192ページ
定価 2,580円(本体)+税
ISBN 978-4-7741-7749-6



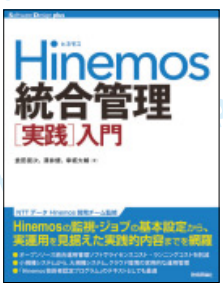
中井悦司 著
B5変形判・200ページ
定価 2,680円(本体)+税
ISBN 978-4-7741-7654-3



上田隆一 著
USP研究所 監修
B5変形判・416ページ
定価 2,980円(本体)+税
ISBN 978-4-7741-7344-3



中島雅弘、富永浩之、
國信真吾、花川直己 著
B5変形判・416ページ
定価 2,980円(本体)+税
ISBN 978-4-7741-7369-6



倉田晃次、澤井健、
幸坂大輔 著
B5変形判・520ページ
定価 3,700円(本体)+税
ISBN 978-4-7741-6984-2



養成読本編集部 編
B5判・176ページ
定価 1,980円(本体)+税
ISBN 978-4-7741-7993-3



養成読本編集部 編
B5判・192ページ
定価2,480円(本体)+税
ISBN 978-4-7741-7858-5



養成読本編集部 編
B5判・128ページ
定価 1,980円(本体)+税
ISBN 978-4-7741-7320-7



養成読本編集部 編
B5判・192ページ
定価 2,280円(本体)+税
ISBN 978-4-7741-7631-4



養成読本編集部 編
B5判・128ページ
定価 1,980円(本体)+税
ISBN 978-4-7741-7607-9

〒162-0846 東京都新宿区市谷左内町21-13
販売促進部 TEL 03-3513-6150 FAX 03-3513-6151

第106回

IT エンジニア必須の 最新用語解説

TEXT：㈲オングス 杉山 貴章 SUGIYAMA Takaaki
takaaki@ongs.co.jp

テーマ募集
本連載では、最近気になる用語など、今後取り上げてほしいテーマを募集しています。sd@gihyo.co.jp 宛にお送りください。

railcar

Rust 実装のコンテナランタイム「railcar」

近年、Dockerを中心としたコンテナ技術のエコシステムは拡大を続けており、大手ベンダーも積極的に対応を進めています。そんな中で、Oracle社は「railcar」と呼ばれる独自のコンテナランタイムをオープンソースソフトウェアとして公開しました。

railcarはOCI（Open Container Initiative）に準拠しており、Rust言語で実装されている点が大きな特徴です。OCIは、Docker社やCoreOS社などが中心となって発足した「The Open Container Initiative」が推進するコンテナラインタイムの標準仕様です。リファレンス実装として「runC」がありますが、そのほかにもCoreOS社やRedHat社など複数のベンダーがOCIに準拠したランタイムをリリースしています。

一方Rustは、Mozilla Foundationが中心となってオープンソースで開発が進められているプログラミング言語です。並列処理をサポートしたマルチパラダイムな言語を目指しており、手続き型や関数型、オブジェクト指向、並列アクターモデルなどのパラダイムを取り込んだ静的型付けの言語として設計されています。

railcarはOCIに準拠したものであるため、機能的にはrunCのような一般的なコンテナランタイムと同等のものが提供されます。ただし、Rustを用いて実装したことによって、ガベージコレクタやマルチスレッドに関するオーバーヘッドを削減しつつ、高いメモリ安全性も併せ持っているという点が大きな強みです。

railcar 開発に至る経緯

コンテナランタイムの標準仕様としてOCIが登場したことによって、各社が自由にOCI準拠のコンテナランタイムを作れるようになりました。このことは、コンテナシステム設計時のランタイムの選択肢が大幅に拡大し、Dockerに依存しないコンテナのライフサイクルを構築できるようになったことを意味します。

その一方でDocker社は、2017年4月に新しいオープンソースプロジェクトとして「Moby Project」をスタートさせました。Moby Projectは、コンテナシステムを複数のコンポーネントの集合体として構築できるようにするものです。コンテナシステムに必要な各種機能はそれぞれ独立したコンポーネントとして提供され、開発者やユーザはそれらのコンポーネントを任意の組み合わせで利用することができます。

このMoby Projectでは、コンテナランタイムの標準仕様としてOCIをサポートしています。したがって、OCIに準拠していればサードパーティ製のランタイムでもDockerのコンポーネントを組み合わせて動作させることができるということです。railcarの開発も、このようなコンテナランタイムの多様化の流れの中で行われました。

Rust が選ばれた理由

Oracle Developers Blogにおける説明では、コンテナランタイムの実装にあえてRustを採用したおもな理由として、メモリ管理の安全性と、名前空間の適切な処理の2つを挙げています。

現状、コンテナユーティリティの実装言語としてはおもにCまたはGoが使われています。しかし、CはLinuxカーネルとのやりとりに優れる反面、メモリ管理に関するセキュリティ上の問題を抱えています。Goについては、開発スピードやメモリの安全性という点では優れている一方で、名前空間の処理についていくつかの制限があるとのことです。

その点、Rustであれば安全なメモリ管理機構を持ち、かつスレッド管理における低レベルの制御も可能なため、名前空間も適切に処理できるというわけです。実際に開発を進めた結果、いくつかの点で解決すべき課題はあるものの、Rustという言語がコンテナユーティリティを実装するうえで優れた選択肢だという十分な手応えを得られたとのことです。

railcarは現時点ではあくまでも実験的な取り組みではあるものの、有用なランタイムの選択肢を増やし、エコシステムの活性化を促進するという意味では、極めて興味深い取り組みと言えるでしょう。SD

GitHub - oracle/railcar
https://github.com/oracle/railcar

DIGITAL GADGET vol.226

安藤 幸央
EXA Corporation
[Twitter] @yukio_andoh
[Web Site] http://www.andoh.org/

コンピュータグラフィックスの祭典SIGGRAPH2017［前編］
～映画の街ロサンジェルス。研究と展示編

CG技術と、その応用

コンピュータグラフィックスとインタラクティブ技術に関する世界最大の学会・展示会である、第44回SIGGRAPH 2017が、7月30日から8月4日の5日間、米国ロサンジェルス・コンベンション・センターで開催されました。

今年は、世界各国から昨年より多い16,500人を超える一般参加者、1,800人を超える発表者の参加がありました。さらに今年は、いくつかの講演内容がYouTube Liveでも配信され、29,000人を超える視聴者がいたそうです。

44回目となる今年のテーマは、“Mind-Body Dualism”。これは日本語に訳すと「心身二元論」という哲学的な用語で、VR映像が流行る昨今、「視覚と身体とは別々でありつつも相関関係がある」という考えが込められています。

今年の基調講演は、伝説のアニメータとして知られ、ドキュメント映画も作られたフロイド・ノーマン氏。今年81歳になる彼を迎えて、対談形式の講演でした。ディズニー初のアフリカ系アメリカ人のアニメータである彼の波瀾万丈な人生、トラブルメーカーとして業界をかき回してきた経歴など、彼が関わってきた『ジャングル・ブック』や『トイストーリー2』の話を中心に紹介されました。また、現在のVRコンテンツの作製者に対して「映像はストーリーが魅力的でなければいけない。テクノロジはあなたを助けてはくれない」という投げかけをしました。

毎年SIGGRAPHでは、CG技術やCGアートへの貢献者を表彰する催しがあり、今年はアーティスト部門でイギリスのアーティスト、Ernest Edmonds氏が受賞。1960年代からFORTRANを使い、今でこそ流行のジェネラティブアートと呼ばれる手法の先駆者です。コンピュータを駆使し、シンプルな構図の作品を数多く輩出してきた功績が認められたものです。また、期待の若手に贈られるSignificant New Researcher Awardには、デジタルファブリケーション（モノ作り）の分野で活躍するBernd Bickel氏、CG界のノーベル賞とも言われるクーンズ賞には、現在ディズニーの研究所でロボティクス分野で活躍されているJessica Hodgins氏、特別な業績が讃えられるAchievement Awardには、コンピュータビジョンの世界では誰もがお世話になっているであろうRamesh Raskar氏が受賞しました。

先進的なアイデアと未来の先取り。CG論文の数々

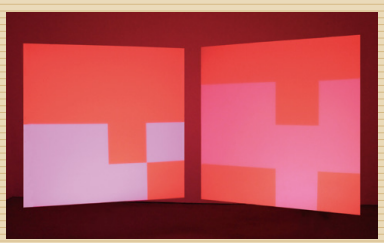
SIGGRAPHの本分は論文発表であり、ここから今年の傾向、業界の流れを読みとることができます。今年



SIGGRAPH会場となったロサンジェルス・コンベンション・センター



伝説のアニメータ、フロイド・ノーマン氏による基調講演の様子



作品名「Shaping Space」というデジタルアート作品

※本記事で紹介しているものは
国内未発表・未発売のものを含んでいます。

のSIGGRAPH論文は、35ヵ国から439本の論文が投稿され、その中の28%である126本の論文が採択されました。論文の分野もCG関連からはもとより、アニメーション、3Dプリンタ活用、画像処理、動画処理、音声処理まで、多岐にわたっています。その中からデジタルガジェット的視点でいくつか紹介しましょう。

SIGGRAPH 2017 発表論文リンク集（非公式版）

※動画やサンプルプログラムへのリンクもあり

http://kesen.realtimerendering.com/sig2017.html

SIGGRAPH 2017 論文集（公式版）

http://www.siggraph.org/sites/default/files/siggraph-2017-papers-open-access.html

Virtual Rephotography: Novel View Prediction Error for 3D Reconstruction

http://www.gcc.tu-darmstadt.de/home/proj/virtual_rephotography/virtual_rephotography.en.jsp

写真をさまざまな方向から何枚も撮影することで、3Dモデルを再現する技術。エラーや間違いの少ない手法（**pic.1**）。

Sequential Line Search for Efficient Visual Design Optimization by Crowds

http://koyama.xyz/project/sequential_line_search/

ビジュアルデザインや設計パラメータの調整で、最良の値を見つけるためにクラウドソーシングを活用する手法（**pic.2**）。

Co-Locating Style-Defining Elements on 3D Shapes

https://www.cs.sfu.ca/~haoz/papers.html

部屋にさまざまな家具を配置する場合に、部屋の雰囲気にあっている形状の家具を集める手法（**pic.3**）。

Retrieval on Parametric Shape Collections

http://cfg.mit.edu/content/retrieval-parametric-shape-collections

好きな形と嫌いな形から、ちょうど良い感じの気に入る形を生成するしくみ（**pic.4**）。

Constrained Palette-Space Exploration

https://www.irit.fr/STORM/constrained-palette-space-exploration/

ユーザが描いた色の雰囲気を保ったままで、色の組み合わせのバリエーションを提案する手法（**pic.5**）。

Playful Palette: An Interactive Parametric Color Mixer for Artists

http://shumash.com/projects

利用しやすいカラーパレットをゼロから作るのはたいへん手間がかかるが、平易にカスタムカラーパレットを作成し利用する手法（**pic.6**）。

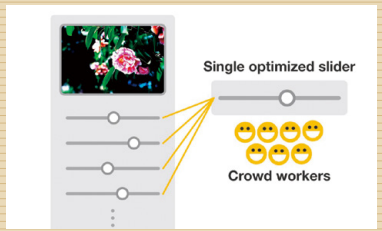
Image-based Reconstruction of Wire Art

http://geometry.cs.ucl.ac.uk/projects/2017/wire-art-reconstruction/

三次元の針金細工で、ある方向からのみ適切に見える形状を作り上げる技法（**pic.7**）。



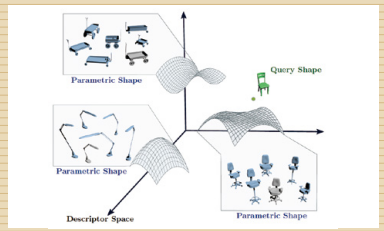
pic.1 Virtual Rephotography: Novel View Prediction Error for 3D Reconstruction



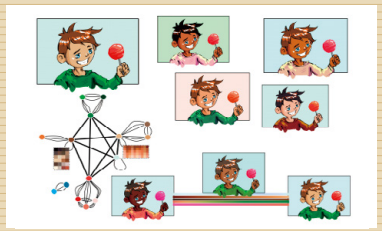
pic.2 Sequential Line Search for Efficient Visual Design Optimization by Crowds



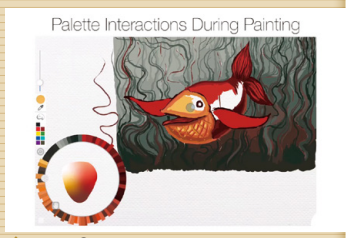
pic.3 Co-Locating Style-Defining Elements on 3D Shapes



pic.4 Retrieval on Parametric Shape Collections



pic.5 Constrained Palette-Space Exploration



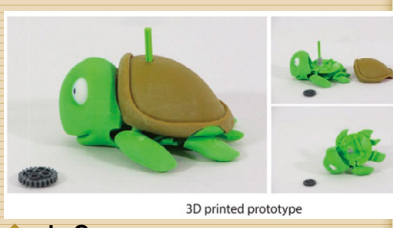
pic.6 Playful Palette: An Interactive Parametric Color Mixer for Artists



pic.7 Image-based Reconstruction of Wire Art



pic.8 Interactive Design of Animated Plushies



pic.9 Functionality-aware Retargeting of Mechanisms to 3D Shapes

Interactive Design of Animated Plushies
http://jamesmbern.com/

縫いぐるみのような布と綿を使ったファブリケーションで動きを再現する、プロトタイプ作成のための技術（**pic.8**）。

Functionality-aware Retargeting of Mechanisms to 3D Shapes
http://visualcomputing.ist.ac.at/publications/2017/MechRet/

カラクリ細工で、実際に動かすことのできる3Dプリントオブジェクトの生成技術（**pic.9**）。

これからのコンピュータグラフィックスの進化

最近のCG技術の傾向としては、機械学習を活用したもの、大量のコンピューティングパワーを活用した研究が盛んになる一方で、クラウドソーシングなどを使って人間の判断やたくさんの人々の意見を集め、それをデジタル的に処理するといった両極端な流れになりつつあります。また、ハリウッド映画で使われる最新の特殊効果から生まれる新しいCG技術もあれば、ビデオチャットやモバイルアプリで使えるような、身近な活用が見込まれる技術も広く研究されるようになっています。

SIGGRAPHで研究発表される技術の数々は机上の空論ではなく、どれも何かのニーズや課題があって、それらを解決するために、新しい手法で、より便利に高速に、安価に精度高く実現できるよう、世界中のさまざまな機関で、さまざまなアプローチから研究されているのが特徴です。

今年の冬、11月27～30日の4日間開催されるSIGGRAPH ASIA 2016は、タイのバンコクでの開催です。また、来年夏のSIGGRAPH 2018は8月12日から16日の5日間、カナダのバンクーバーで開催されます。さらに、来年冬のSIGGRAPH ASIAは東京での開催が予定されています。次回のSIGGRAPHでも、さらに新しい技術とアートの共創が期待されます。SD

Gadget 1

» PKEメーター

※イベント展示品のため参照URLなし

映画「ゴーストバスターズ」に登場した幽霊探知機

2016年に公開されたリメイク版映画「ゴーストバスターズ」に登場した、ラウンチャー、銃、幽霊探知機（下段一番右）などの展示。最新の特殊効果だらけの映画でも、実際に俳優が手にする小道具は、ごく普通の工作によって作られた感じのするオモチャのような物でした。実際の家電製品やモデルガンなどの部品を切り貼りして作られているそう。SIGGRAPH会場のプロダクションギャラリーというコーナーで展示されていました。



Gadget 2

» 赤外線ゴーグル

※イベント展示品のため参照URLなし

映画「インビジブル」に登場した赤外線ゴーグル

透明人間を描いた2000年公開の映画「インビジブル（原題 Hollow Man）」で使われたヘルメット（上段一番左）、マスク（上段左から2番目）、赤外線ゴーグル（下段左から2番目）などの展示。下段右にあるのは2012年公開の「メンインブラック3」に登場した銃器とIDカード。記憶を消してしまう装置、ニューラライザーも展示されているハズなのですが、写真には見当たりませんね。見た記憶を消されてしまったのでしょうか……。同じくプロダクションギャラリーでの展示。



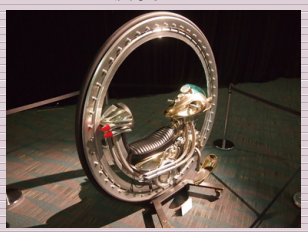
Gadget 3

» モノバイク

※イベント展示品のため参照URLなし

映画「メンインブラック3」に登場したモノバイク

2012年の映画「メンインブラック3」に登場した、一輪で動くモノバイクの実物大模型の展示。映画のシーンでは、壊れた車に搭載されているモノバイクを引っ張り出して乗りこなしていました。実際の映像のほとんどのシーンはCGで作られたモノバイクだそうですが、乗り込むシーンなど、一部ではこの1/1模型が使われたもよう。とてもリアルな質感ですが、残念ながら実際にはまったく動けないそうです。同じくプロダクションギャラリーでの展示。



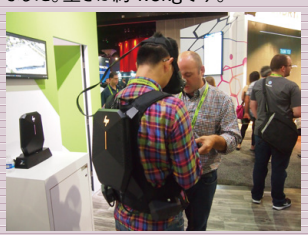
Gadget 4

» HP Z VR Backpack PC

http://www8.hp.com/us/en/campaigns/vrbackpack/overview.html

バックパック型VRパソコン

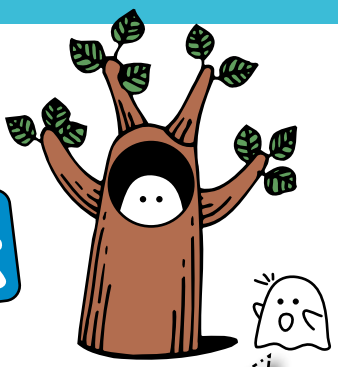
ヒューレットパッカード社が発売を予定しているVR専用のバックパック（リュック）型のハイパワーパソコン。VRヘッドセットをかぶって動き回る際、処理用のパソコンから長く伸びたケーブルを引きずり回すのは面倒で危険なため、背中に背負えるサイズと形状を持ち、かつVR処理のために高い処理能力を両立したパソコンです。現在はまだ未発売ですが、会場では実際に動作しているマシンがデモ公開されていました。重さは約4.6kgです。



結城 浩 YUKI Hiroshi
http://www.hyuki.com/
Twitter：@hyuki

53

結城浩の再発見の発想法



Checksum ―― チェックサム

チェックサムとは

チェックサム（Checksum）とは、データから生成される、誤り検出のための小さな数です。チェックサムはサムチェックと呼ばれることもあります。チェックサムは、ファイルが壊れていないかどうか、あるいはネットワークで受信したデータが壊れていないかどうかを確かめるときによく使われます。

UNIXにはcksumというコマンドがあり、ファイル名を指定して実行すると、以下のようにチェックサム、バイト数、ファイル名が表示されます。

```
$ cksum important.pdf
41758665 203529795 important.pdf
```

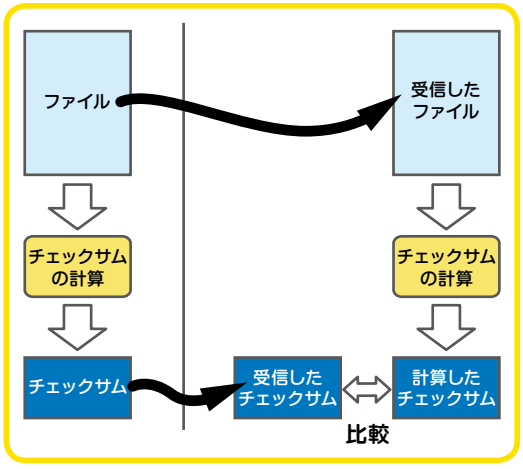
このファイルimportant.pdfは203,529,795バイトもある大きなものですが、cksumはその内容を41758665というチェックサムの値に要約しているのです。チェックサムの値をのちほど再計算したときに、41758665以外の値に変化していたら、important.pdfのファイルの内容が変化していることがわかることになります。

ファイルの内容が変化したかどうかを確認するだけなら、important.pdfを別ファイルbackup.pdfにコピーしておいて、ファイルを直接比較するという方法もあるでしょう。しかし、ファイルの大きさが大きい場合には比較のための別ファイルを保存しておく場所が問題になりますし、important.pdfを他人から受信したとき、このファイルが他人が持っているものと同じかどうかの判定は難しくなります。でも、ファイルと一緒にチェックサムの値も受信すれば、自分の手元でチェックサムを再計算して比較することで、ファイルが壊れていないか調べることができるでしょう（**図1**）。

ここでちょっと注意が必要です。チェックサムが違っていたら、確信を持って「誤りが存在する」と言えます。しかし、チェックサムが同じままであっても、誤りが存在する可能性は残ります。誤りをどれだけ正しく見つけるかの能力（誤り検出力）はアルゴリズムによって異なります。

また「誤り検出」と「誤り訂正」の区別も大事です。チェックサムが違っていたら、誤りが存在したことはわかります。これは「誤り検出」です。でも、誤りが存在していることはわかっても、

▼図1　チェックサム



正しいデータがわかるわけではありません。ですから、チェックサムで「誤り訂正」はできません。

チェックサムのアルゴリズム

データから小さな数を生成させて誤りの検出に使うアルゴリズムは多数あります。

もっとも単純なのは**すべてのデータを順番に加えていき、最後に得られた和を適当な数で割った余りをチェックサムにする**というアルゴリズムでしょう。たとえば、データを1バイトずつ加えていき、和の最下位となる1バイトをチェックサムにするアルゴリズムは、どんなプログラミング言語でも簡単に実装できますね。もともとチェックサムという名前はこのアルゴリズムからきています(サムは「和」の意味)。このアルゴリズムは単純でそれなりに有効ですが、データの中にいくら0が挿入されてもチェックサムが変化しない点や、データ中でバイト単位の入れ換えが起きてもチェックサムが変化しないという欠点があります。

通信でよく使われるのが**CRCチェックサム**です。CRCはCyclic Redundancy Code/Checkの略で、巡回符号の理論を背景に誤り検出力を高めているアルゴリズムです。実際には単純な「和」ではないのですが、慣用的にCRCチェックサムと呼ばれることがあります。cksumコマンドでもCRCチェックサムの一種が使われています。

暗号技術でよく使われるのは**メッセージダイジェスト**や**一方向ハッシュ関数**と呼ばれるアルゴリズムです。これは悪意のある第三者が意図的に改ざんを行おうとしても、改ざんを検出することができるという性質を持っているアルゴリズムです。具体的にはSHA-256やSHA-3などが使われます。

日常生活とチェックサム

チェックサムを計算するのはコンピュータのプログラムの仕事ですが、**データそのものを比べるのではなく、別の情報を比べて誤りを検出する**という発想は、日常生活のあちこちに登場します。

日常生活で誤りを検出するときに、**個数**を確認するのはよく使われる方法です。たとえば、文具店でハサミと付箋紙(ふせんし)とノートと万年筆と消しゴムと……とにかく多種類のものを買うとしましょう。購入漏れがないかどうかを確認するため、私たちはよく「個数」を数えます。ぜんぶで10個の品物を購入するはずだったのに、カゴに入っているのが9個しかないとしたら、何が足りないかはまだわかりませんが、何かが足りないことは確実です。つまり、誤りの検出ができたことになりますね。検出ができれば、実際には何を買い忘れたのだろう？と調べ始めることができます。

長い数字列の一部を使って誤りを検出することも可能です。たとえば、銀行口座を複数持っていて用途ごとに分けている場合、銀行口座の口座番号をすべて暗記することは現実的には難しいでしょう。でも、末尾の3桁を覚えておくことはそれほど難しくはありません。口座をまちがえないようにするため、末尾の3桁を使って誤り検出するというのは現実的に有効です。

書籍の番号として使われているISBN-13(International Standard Book Number)の末尾の1文字は**チェックディジット**と呼ばれています。チェックディジットは最初の12桁の数字から計算で得られますので、入力の誤りを検出することが期待できます。入力の誤りを確かめるのに、入力したISBNそのものを確かめるのではなく、別途計算したチェックディジットで確かめるのがポイントです。

◆ ◆ ◆

あなたの周りを見回して、「何か」に誤りがないかチェックする状況を探してください。そのとき、「何か」を直接チェックしているでしょうか。それともチェックサムのように別のルートで得た情報でチェックしているでしょうか。また、誤りの「検出」と「修正」を区別しているでしょうか。

ぜひ、考えてみてください。SD



及川卓也の プロダクト開発の道しるべ

品質を高めるプロダクトマネージャーの仕事とは？

第12回 プロダクトマネージャーの実務

Author 及川 卓也（おいかわ たくや）
Twitter @takoratta

すでに、この連載も12回目となりました。

さて、過去11回でプロダクトマネージャーに必要なスキルや方法論などを解説してきましたが、実際にプロダクトの開発サイクルの各フェーズでどのようにプロダクトマネージャーがかかわるかはまだ解説していませんでした。プロダクトマネージャーの役割も組織によって少しずつ異なるのと同じように、プロダクトマネージャーの実務作業も組織ごとに違います。ですが、多くの組織で共通に行っている作業があるのも事実です。今回はそのプロダクトマネージャーの実務について説明したいと思います。



プロダクトマネージャーの役割

本題に入る前に、プロダクトマネージャーの役割をおさらいしてみましょう。

プロダクトマネージャーはプロダクトチームをリードし、プロダクトを成功に導く役割を担います。プロダクトチームとは、エンジニアやデザイナー、QA（Quality Assurance：品質保証）担当者などの開発に直接携わる人だけでなく、マーケティングや広報、営業、カスタマーサポート、法務担当者なども含まれる、組織横断的なチームです。もちろん、専属でない人も多くいますが、そのような人たちも含めて、プロダクトの成功を考える人たちを率いるのがプロダクトマネージャーです（**図1**）。



プロダクト開発に必要なタスク

このような役割を持つプロダクトマネージャーのプロダクト開発の各フェーズにおける実務をさっそく見てみましょう。

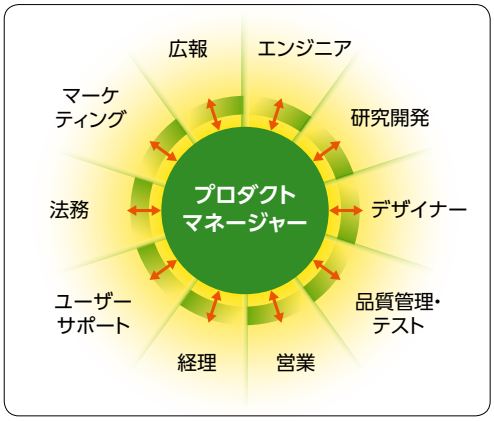
プロダクトの開発フェーズはいくつかの分け方があると思いますが、ここでは**図2**のように企業や組織のビジョンやミッションから落とし込んでいく方法をとってみましょう。



ビジョンとミッション

すべての企業や組織にはその存在意義があります。その存在意義、すなわち、なんのために事業を行うかを説明したものがミッションです。より具体的には、企業や組織の事業内容と目的を整理し、どのようにその事業目的にたどり着

▼図1　複数のチームメンバーから構成されるプロダクトチームとPM



くのかを定義しています。

一方、ビジョンは企業や組織が長期的にどのような姿となることを目指すかを定義したものです。

プロダクトマネージャーが企業や組織全体のビジョンやミッションの策定に関わることは稀かもしれません。しかし、もしあなたが起業間もないスタートアップの一員ならば、ビジョンやミッションを創業者たちと一緒に考えることもあるでしょう。そして、何よりも大事なのは、その企業のビジョンやミッションを基にして、あなたが担当するプロダクトのビジョンやミッションを考えることです。

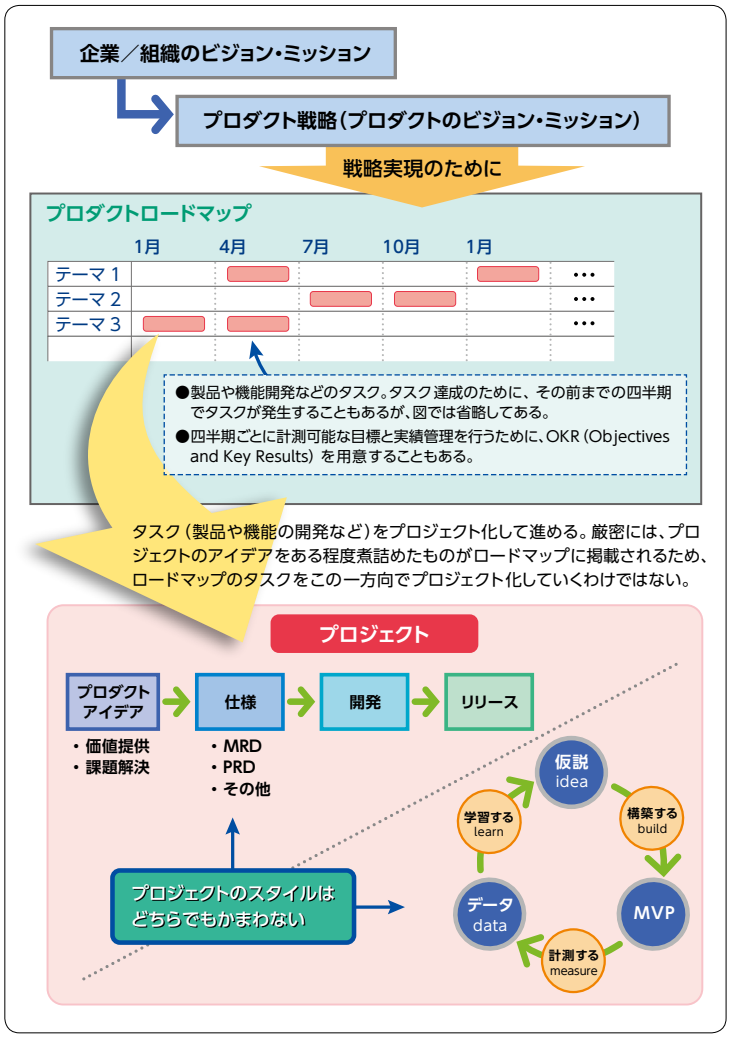
プロダクトの戦略としては、ビジョンやミッション以外にもさまざまなことを考えないといけませんが、もっとも大事なのは、企業がそうであるように、その存在意義であるビジョンとミッションを考えることです。プロダクトが何のために開発されるのかがプロダクトのミッションで、長期的にそれが社会にもたらす影響などを考えるのがビジョンです。

プロダクトマネージャーに大事なこととして、「ぶれないこと」を筆者はいつも主張しますが、ぶれないためにプロダクトにとっても大事なのはビジョンとミッションです。

ビジョンやミッションの策定方法はいくつかありますが、この連載でも紹介したインセプションデッキのエレベーターピッチ(**図3**)やリーンキャンバスを使うのでも良いでしょう。

プロダクト戦略と言った場合、ビジョンやミッションだけでなく、意思決定のためのフレームワークや目的実現のための施策なども含まれます。企業のビジョンやミッションとの整合性と目的実現のために必要なリソースなどもすべて議論し、それこそ「ぶれなく」プロダクト開発を進められるようにしなければなりません。

▼図2　プロダクトの開発フェーズ。企業／組織のビジョンやミッションから、プロダクト戦略を決定し、それを基にプロダクトロードマップを起こし、1つ1つのタスクを実施していく



プロダクトロードマップ

プロダクト戦略が明確になったら、それを実現するための道筋であるロードマップを策定します。ロードマップには、何をいつするかを記載していきますが、筆者はその前にプロダクトの柱と

▼図3　インセプションデッキのエレベーターピッチのフォーマット

- [潜在的なニーズを満たしたり、抱えている課題を解決したり]したい
- [対象顧客]向けの、
- [プロダクト名]というプロダクトは、
- [プロダクトのカテゴリ]である。
- これは[重要な利点、対価に見合う説得力のある理由]ができ、
- [代替手段の最右翼]とは違って、
- [差別化の決定的な特徴]が備わっている。

なるテーマを決めることをお勧めします。テーマとは、開発タスクのグループです。これ自体はなんと呼んでもらってもかまわないのですが、タスクの共通テーマと考えるほうが自然なので、筆者はテーマと呼んでいます。

これはたとえば、BtoCのWebサービスがプロダクトだとしたならば、「新規ユーザ獲得」や「日次アクティブユーザ数（DAU：Daily Active Users）増加」などがテーマとなり得ます。この例のように、KPI（Key Performance Indicator）がテーマとなることも多いです。

テーマが決まったら、それぞれのテーマごとにタスクを決めます。タスクは多くの場合は新機能開発であったり、既存機能の改善です。ロードマップには、そのタスクをいつ行うかを記載していきますが、ただ単に思いつきをロードマップに入れるのではなく、誰のために、なぜ行うのかをしっかり検討したうえで記載していきます。また、タスク間には依存関係があったり、タスクを実現するためのリソース（多くの場合は人的リソース）が他タスクと取り合いになることもあります。ロードマップの時点では厳密なプロジェクト計画まで落とし込む必要はありませんが、実現不可能な予定とならないように気をつける必要があります。



プロジェクト計画

ロードマップに記載されたタスクはプロジェクトとして実施していきます。プロジェクトという呼称にこだわる必要はありませんが、ここでは便宜上プロジェクトと呼んでいます[注1]。タスクとなる新規開発や機能改善は、新たな価値提供か課題解決のいずれかを目的とします。したがって、ここで大事なのは、価値仮説や課題仮説を検証することです。多くの場合、アイデアは価値や課題と、その提供方法や解決方法がまとめられたものです。この場合、最初に行うのは、それぞれを分離し、価値や課題が本当に求められているものかの検証です。次に、その提供方法や解決方法がその価値や課題に対して最適なものかどうかの検証です。

図2では、タスクからプロダクトアイデアを検討するように描いていますが、実はタスクを考える際に、プロダクトアイデアの検討なくしてはタスクは定義できません。したがって、ロードマップ策定時にある程度のアイデアの議論はされていることが一般的です。

アイデアの検証ができたならば、それを仕様に落とし、開発を進めていくことになります。**図2**では、アイデアから仕様を決定し、開発し、リリースするというフローと、リーン開発として構築→計測→学習という進め方の両方を併記しています。図の上では、ウォータフォールとアジャイルの対比のように見えるかもしれませんが、必ずしもそれを意図したものではありません。開発プロセスは何であれ、プロダクトマネージャーはアイデアの生成と検証、仕様策定、開発時の意思決定、リリース準備やリリース後の対応を行っていくことになります。



プロダクトマネージャーの仕事

説明したプロダクト開発の各フェーズでのプロダクトマネージャーの仕事は自明な部分も多いと思います。プロダクト戦略やプロダクトロードマップの策定はもちろんのこと、もし企業が事業目標と実績を四半期ごとにまとめるOKR（Objectives and Key Results）[注2]を採用してい

注1）プロジェクトは開始と終了が明確になっていることがほとんどですので、イテレーションを繰り返し、完成度を高めていくリーン開発の場合はプロジェクトという呼称はあまり相応しくないかもしれません。

注2）米国のハイテク企業を中心に行われている事業目標と実績の管理方法。

るようでしたら、プロダクトOKRの策定、PRD（Products Requirement Document）を始めとする仕様や各種検証などがプロダクトマネージャーの仕事となります。

また、すべてのフェーズを通じて、アイデアマネージメント、タスクの優先度付け、チームマネージメントなどの仕事がプロダクトマネージャーに期待されます。チームマネージメントにはメンバーのモチベーション管理や各種意思決定の場でのファシリテーションなども含まれます。

よく、プロダクトマネージャーはエンジニアが開発に入ったら、何をやっているのかわからないと言われることがあるのですが、仕様の確認作業は常に発生しますし、優先度付けなどの意思決定が必要な場面は日常的です。また、リーン開発の場合に限らず、ユーザからのフィードバックを確認し、開発のイテレーションに回すことも必要です。

◆ ◆ ◆

以上、今回はプロダクトマネージャーの実務として、日々プロダクトマネージャーが何をしているかの解説を、企業のミッションとビジョンから俯瞰（ふかん）する形で解説しました。次回、さらに各作業について解説をする予定です。SD

column ハードウェア製造における PRD（Product Requirements Document）

何（What）をどうして（Why）作るのかを明確にするのが役割のプロダクトマネージャーの重要なツールがPRD（Products Requirement Document）です。この連載では、おもにソフトウェア開発を想定した形でPRDの解説をしましたが、PRDはソフトウェアだけでなく、ハードウェアの開発においても重要な役割を果たします。その良い例が筆者が参加したカンファレンス、builderscon で発表されていました。

builderscon[注3]は「『知らなかった、を聞く』をテーマとした技術を愛する全てのギーク達のお祭り」で、今年は8月4日と5日に慶應大学日吉キャンパスにて行われました。

その2日目に行われたFactory Classというセッションは、タイトルからデザインパターンかなにかの話かと思ったのですが、まったく違い、もともとソフトウェアエンジニアだったスピーカー（Jesse Vincent氏）がキーボード製造を行うまでの経験を披露するものでした[注4]。

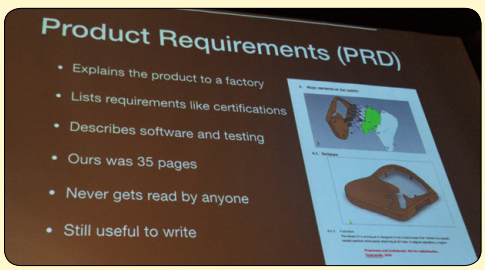
Vincent氏は製造を行う工場を中国や台湾で探すのですが、交渉は一筋縄ではいかず、何度もやり直しを強いられます。その中のエピソードとして話されていたのが、PRDの重要性です。

工場とのやりとりで苦労したのが、オーダーする内容をいかに曖昧さを排除した形で伝えるかでした。依頼に対しての見積もりにしても、工場によってピンキリだったり、製造過程もそれぞれの工場によって異なったりしたそうです。そんな中、PRDが製造するものすべてを網羅的に説明する唯一のドキュメントとして重宝した、そんなエピソードを披露していました（**写真A**）。

ただし、35ページにものぼるPRDは、依頼をかけた工場側には一切読まれることはなかったそうです。この事実を伝えられた会場は笑いに包まれましたが、Vincent氏は「それでも一番役に立ったドキュメントだった」と話しました。

この事実は何を示唆するでしょうか。1つはPRDは顧客のためではなく、自分たちのために、自分たちが作ろうとしているものを明確に説明するために役立つものであるということ。もう1つはPRDはやはり重厚長大になりがちで、読まれない可能性があるということ。後者に関しては、この連載でも紹介したような、OnePagerと呼ばれる短い説明や、インセプションデッキなどを用意するようにして補完することが大事であることがわかります。

▼写真A　Jesse Vincent氏の発表より



注3）https://builderscon.io/

注4）そのキーボードはhttp://keyboard.io/に詳しく紹介されています。Kickstarterでの購入も可能です。

Profile
ソフトウェアエンジニアとして社会人キャリアをスタートした後、MicrosoftやGoogleでプロダクトマネージャーやエンジニアリングマネージャーを経験。現在はいくつかのスタートアップのプロダクトマネージメントをサポートしている。

宮原徹の

オープンソース放浪記



第20回 今年のOSC京都も暑かった!

宮原 徹(みやはら とおる) Twitter @tmiyahar 株式会社びぎねっと

猛暑のOSC京都

オープンソースカンファレンス京都は毎回、夏に開催されます。今年も8月4日(金)、5日(土)という夏真っ盛りの時期に開催されました。この時期は台風5号の影響で西日本は猛暑に襲われており、8月6日(日)には島根県益田市で39.3度の気温を観測。もちろん、京都も暑かった! そんな京都の様子をレポートします。

今年も京都リサーチパークで開催

11回目の開催を迎えるOSC京都の会場は、ここ数年、京都リサーチパーク(KRP)で開催しています。市内の中心部からは若干外れますが、少し南に下ると京都鉄道博物館や京都水族館がある、やや西側のエリアです。毎回たくさんの出展やセミナーが開催され、来場者も多く訪れるため会場探しには苦労するOSCですが、KRPは展示スペース、セミナー会場ともに豊富なため、OSC京都の開催が成り立っています。また、この期間は「KRP WEEK」という会場全体でのイベントが行われていることもあって、会場側からのOSCへのバックアップがあるのもうれしいところです。東京などではなかなかこのような会場がないため、毎回たいへん助かっています。

▼写真1 懇親会でスタッフ集合写真。今回もたくさんのスタッフが開催を支えてくれました



▼写真3 納涼床から鴨川を眺めて。夕暮れの空と涼しい風が楽しめました



OSC京都は学生スタッフが大活躍

OSC京都の特徴の1つとして、学生スタッフが運営に大活躍してくれている点が挙げられます。前日準備での配付資料の袋詰めや、2日間の受付、会場運営など、規模の大きいイベントでは多くのスタッフが必要となります。今回は京都女子大学、そして大阪のECCコンピュータ専門学校の学生さんたちがスタッフとして参加してくれました。また、京都開催のもう1つの特徴として、学生出展が多いことも挙げられるでしょ

▼写真2 京都といえばおばんざい。これが冷酒に合います



う。今回も普段の活動を紹介する展示ブースに、学生さんの説明員がたくさん立って盛り上がっていました。こういうところは、まさにOSCの「OSSの文化祭」というキャッチフレーズにピッタリですね。実際、終了後の懇親会(**写真1**)も、学生さんがそのまま参加するため、ほかの地域の懇親会に比べると「若い人が多いな」と感じます。ほかでの開催も、どんどんこういう雰囲気を作っていきたいですね。

京都の夜は食べ歩き、飲み歩き

OSC京都のもう1つの魅力は、夜の食べ歩き、飲み歩きです。ここ最近では、前日の準備が終わったあとの夜の食事は五条烏丸にある「蕎麦の実 よしむら」というお店に決まりつつあります。店名のとおり、お蕎麦をメインにした居酒屋という感じですが、いろいろな蕎麦料理が楽しめます(**写真2**)。それに合わせた日本酒も各種取りそろえていますが、ここはやはり地元京都の日本酒でしょうか。

それ以外によく行くのが、鴨川の西側にある細い路地にお店がたくさん並んでいる先斗町(ぽんとちょう)です。この時期は川岸の店から川面の空中にせり出した「納涼床」が楽しめます(**写真3**)。今回も納涼床で日本酒と串揚げを堪能しましたが、日中の暑さに比べて、鴨川を抜ける風が涼しかったのが印象的でした。こういったその地域ならではの楽しみ方ができるので、暑い京都でのOSC開催も楽しめるというものです。

イギリスからPimoroniの人たちがやってきた

場所は変わって東京でのイベントの話題です。最近、Raspberry Piでいろいろとやることが多くなってきているのですが、8月8日(火)の夜にRaspberry Piを使って電子工作をする人たちの集まりである「Raspberry Jam Tokyo #008」に参加してきました。普通の勉強会と違って、自分の作ってきた電子工作をテーブルの上に並べて、お互いにああだこうだと情報交換をする楽しい会でした。今回はイギリスでいろいろなRaspberry Pi関係のハードウェア製品を開発・販売しているPimoroni社から、Paul Beech氏とNiko Kotilainen氏が来日してゲスト参加してくれたこともあり、なかなかの盛り上がりを見せました。イベント自体もビール片手に、という感じでしたが、終了後はさらにスタッフの打ち上げがあり、さらに大盛り上がりしました(**写真4**)。こんなふうに交流ができるのも、オープンなイベントならではですね。SD

▼**写真4　Pimoroniのみなさんと記念撮影。(飲み過ぎで)大盛り上がりです**





Report

細い路地にこそ名店あり?

隠れた名店を探すのも、これまた地方出張の楽しみです。今回も納涼床での食事を終わらせた後(観光地価格ですのでちょっと高いのです)、いい感じのお店を探します。いろいろと彷徨(さまよ)った挙げ句、細い路地を発見しました。経験上、こういう細い路地を入ったところには、こぢんまりとしていながらもこだわりのお店があるものです。一般人なら躊躇(ちゅうちょ)するような路地ですが、気にせず入り込んで店をチェックしていくと、日本酒BARを発見。ここはあやしい、ということで突撃したところ、これが大当たりでした。このお店、珍しい日本酒が100種類以上あるがメニューはないので、適当に頼むと店主が選んで出してくれる、というシステム。3種類飲み比べなどはあるのですが、金額も時価という、日本酒が好きな人でないと困ってしまうようなお店でした。お店の名前はあえて伏せておきますので、みなさんも京都に行ったら細い路地を探してみてください。私は次に京都に行ったら、多分また行ってしまうと思います。



▶**島根の隠れた銘酒「王禄」の20年古酒という謎の逸品。なんでこんなものがあるのでしょう?**

ツボイのなんでもネットにつなげちまえ道場

第28回

BBC micro:bitを使ってみる

Author 坪井 義浩（つぼい よしひろ） Mail ytsuboi@gmail.com @ytsuboi

はじめに

この連載の第14回（2016年8月号）で紹介したBBC micro:bit（**写真1**）ですが、あれから約1年、工事設計認証（いわゆる技適）を受けて日本で適法に利用できる状態で上陸してきました！　去る8月5日、Maker Faire TokyoでMicro:bit財団のCEOであるZach Shelby氏が基調講演を行いました（**写真2**）。今回は、前回は概論しか説明できなかったmicro:bitを実際に使ってみたいと思います。

昨年からの変化

昨年は、「ブロックをつなげてプログラミングを行うBlock Editor、Microsoft Researchが提供するタッチデバイス向け開発環境のTouch Develop、Code KindomsのJavaScript、Python、とさまざまな選択肢が提供されています」とmicro:bitの開発環境を紹介しましたが、現在ではJavaScript Blocks EditorとPython Editorの2種類[注1]に整理されました。

ブロックでプログラミング

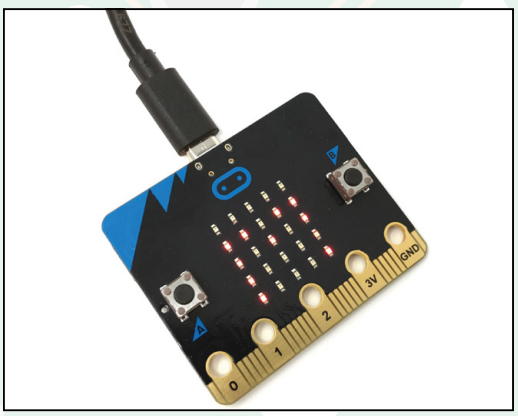
では、micro:bitの特徴の1つである、ブロックの組み合わせによるプログラミングをしてみましょう。しかし、単にmicro:bitの5×5ドットのLEDに何か表示してみるだけでは芸がありませんので、ここは何かセンサなどをmicro:bitに接続してコントロールしてみたいと思います。

micro:bitは、第14回（2016年8月号）でも紹介したように、基板の端にあるカードエッジコネクタを使ってほかの機器と電気的に接続できます。今回は、Grove Inventor Kit for micro:bit[注2]（**写真3**）という、micro:bitをGroveと接続できるキットを使ってみます。このキットには、JavaScript Blocks Editorを使ってキットに同梱のGroveモジュールを活用するチュートリア

注1）http://microbit.org/code/

注2）https://www.seeedstudio.com/Grove-Inventor-Kit-for-micro%3Abit-p-2891.html

▼写真1　BBC micro:bit



▼写真2　Zach氏（中央）と筆者（右）



ルが提供されています。

このチュートリアルのステップ8に、超音波距離センサで測った距離を4桁の表示機に表示するものがあります。今回はこれをやってみましょう。まず、キットの中に入っている部品を**図1**のように接続します。

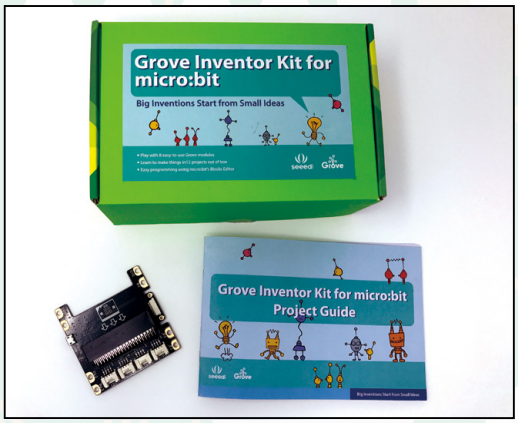
次にJavaScript Blocks Editor[注3]にアクセスし、まずGroveのモジュールを使うためのパッケージ(ブロック)をエディタに追加しましょう。エディタを開くと(**図2**)、「高度なブロック」という項目があります。ここをクリックして開き、「パッケージを追加する」をクリックします。するとダイアログが表示されますので、入力欄に「`https://github.com/Seeed-Studio/pxt-grove`」とURLを入力して虫眼鏡のアイコンをクリックしてください。するとGroveパッケージがダウンロードされ、「計算」の下に追加されます。こうしてパッケージを追加することで、Groveモジュールを使うためのブロックが追加されます(エディタの言語は、Microsoftロゴの左にある歯車のアイコンをクリックし、「Language」を選択すると日本語に変えることができます)。

必要なブロックが使えるようになったところで、ブロックを**図3**のように組み合わせてプログラムを作ります。ブロックを組み合わせ終えたところで、ダウンロードボタンをクリックすると、プログラムがビルドされ、ブラウザが自動的にmicro:bitに書き込むためのHEXファイルのダウンロードを始めます。

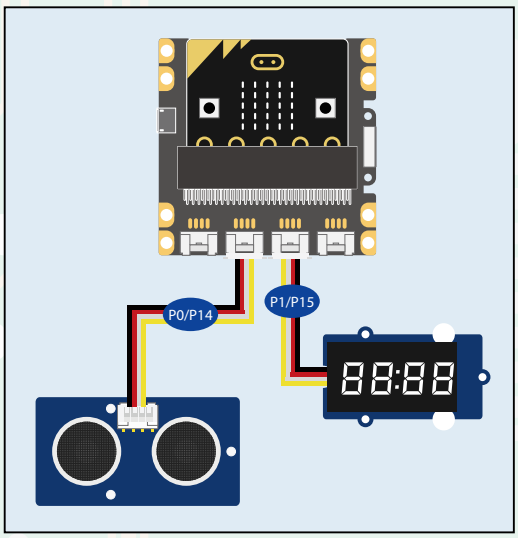
あとは、micro:bitをパソコンにUSBケーブルで接続し(micro:bit本体のUSBポートと接続してください)、パソコンでマウントしたドライブにHEXファイルを書き込

注3) https://makecode.microbit.org/

▼写真3　Grove Inventor Kit for micro:bit



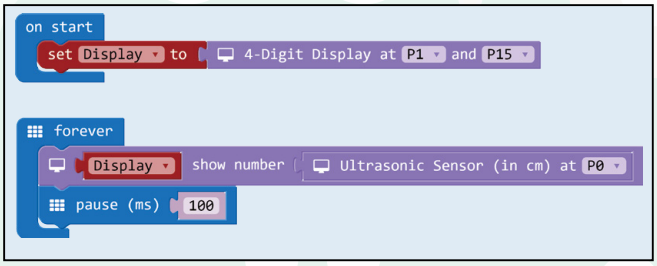
▼図1　Groveの接続方法



▼図2　JavaScript Blocks Editorの画面



▼図3　ブロックの組み立て



むだけです。書き込みが終わると、**写真4**のように4桁のLED表示機に超音波センサで測った距離が表示されます。

ちなみに、このHEXファイルは、エディタの「プロジェクト」→「ファイルを読み込む」でアップロードすることによってブロックに戻すことができます。筆者のHEXファイルをGist[注4]に置いておきましたので、ブロックを組み合わせるのが面倒な方は、このHEXファイルをダウンロードして読み込んでみてください。

Mbedでやってみる

ブロックを組み合わせたプログラミングは入門者には手軽ですが、本誌の読者には少々物足りないでしょう。ということで、Arm Mbedで同じようなプログラムを組んでみたいと思います。

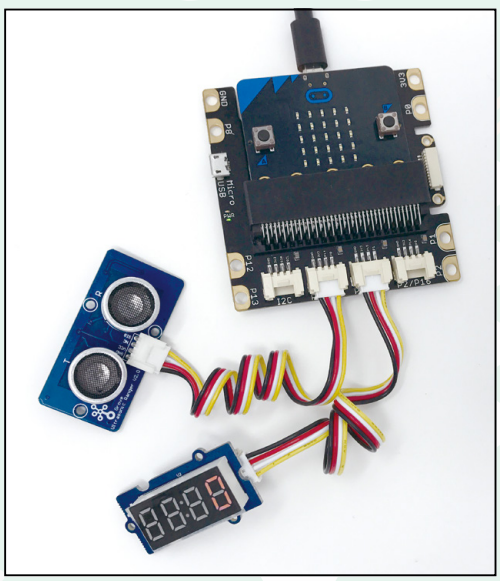
今回から「Arm Mbed」の表記が替わったことにお気づきでしょうか。8月1日にアームはロゴを刷新し、同時にMbedのロゴも新しくなりました。テキスト表記の際には、この「Arm Mbed」ということですので、今回から本稿でも表記を変更しています。新しくなったロゴはMbedのWebサイト[注5]を見てみてください。

さて、Mbedのオンラインコンパイラで開発をするには、まずプラットフォームをオンラインコンパイラに追加しなければなりません。BBC micro:bitのプラットフォームページ[注6]にアクセスし、「Add to your Mbed Compiler」ボタンをクリックして追加します。Mbedのアカウントを持っていなければ、ここで作ってください。

次に、このページにある「microbit-component-display」というサンプルプログラムをインポートします。このとき、ダイアログの「Update all libraries to the latest revision（すべてのライブラリを最新版にする）」にチェックを入れておきましょう（**図4**）。次に、超音波距離センサのライブラリ[注7]を、「Import library」ボタンをクリックし、オンラインコンパイラでTarget Pathを

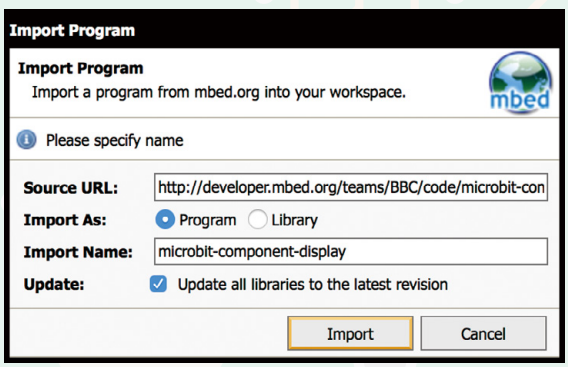
注4）https://gist.github.com/ytsuboi/0d9cc14cd8f8fac2748f2ec1948cf1ea

▼写真4　動かしてみたところ



注5）https://developer.mbed.org/
注6）https://developer.mbed.org/platforms/Microbit/
注7）https://developer.mbed.org/components/Grove-Ultrasonic-Ranger/

▼図4　プログラムのインポート



「micro bit-component-display」にしてImportボタンをクリックしてインポートします。このときダイアログの「Update all sub-libraries to the latest revision(すべてのサブライブラリを最新版にする)」にチェックを入れておきます(**図5**)。同様に、4桁ディスプレイのライブラリ[注8]も、プログラムにインポートして追加します。あとはコードを書くだけです(最新版にすることを忘れずに!)。

コードは**リスト1**のようにとてもシンプルですが、ピンの指定方法で少々ハマりました。ピンを指定するための名前は、Mbed OSのソースコード[注9]の当該部分を参照して調べました。

main.cppも記述できたところで、オンラインコンパイラ画面右上のターゲットが「BBC micro:bit」になっていることを確認し、「コンパイル」ボタンをクリックしましょう。ビルドが行われ、ブラウザがHEXファイルをダウンロードします。このHEXファイルをmicro:bitをパソコンに繋いで認識されているドライブにドラッグ&ドロップして書き込めば動き始めます。こうして作ったプログラムは筆者が共有しておきましたので[注10]、これを「import into Compiler」ボタンをクリックするだけで自身のオンラインコンパイラに利用できます。

先ほどJavaScript Blocks Editorで作ったプログラムとは少し挙動が違い、桁がゼロ埋めされていますし、更新頻度もMbed版では1秒にしたために少なくなっています。どうしても同じ挙動にしたい方は、ちょっと頑張ってみてください。

なお、Mbedは、Mbed OS 5が主流になっていますが、BBC micro:bitは以前のOS 2のみでサポートされています。これは、micro:bitに搭載されているnRF51822というチップのメモリの容量の都合で、OS 5で標準的に使用されているRTOSを動かすのが困難なためだと考えられます。

注8) https://developer.mbed.org/components/Grove-4-Digit-Display/

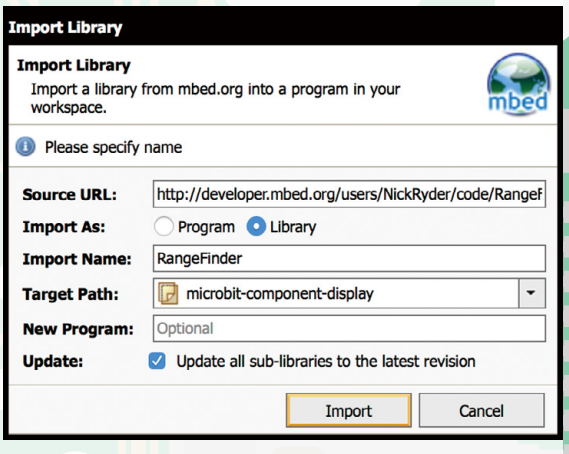
注9) https://github.com/ARMmbed/mbed-os/blob/master/targets/TARGET_NORDIC/TARGET_MCU_NRF51822/TARGET_NRF51_MICROBIT/PinNames.h#L66-L85

注10) https://developer.mbed.org/users/ytsuboi/code/microbit-8-ultrasonic-meter/

まとめ

このように、大人も子供も遊べるmicro:bitはなかなか優れたプラットフォームだと思います。この記事が出るころには夏休みはとうに終わってしまっていますが、micro:bitで遊んでみてはいかがでしょうか。SD

▼図5　ライブラリのインポート



▼リスト1　main.cpp

```
#include "MicroBit.h"
#include "DigitDisplay.h"
#include "RangeFinder.h"

MicroBitDisplay display;
DigitDisplay digit(P0_2, P0_21); // P1/P15

RangeFinder rf(P0_3, 10, 5800.0, 100000);

int main()
{
    float d;

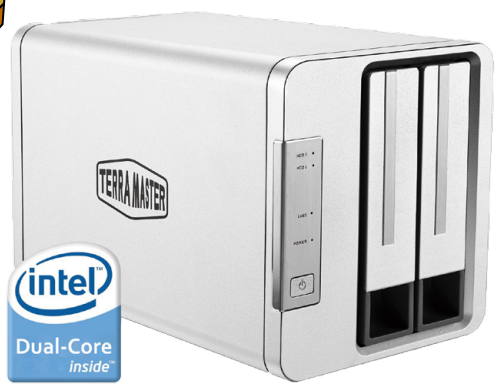
    while(1) {
        d = rf.read_m() * 100;
        digit.write(0,(int)d / 1000 % 10);
        digit.write(1,(int)d / 100 % 10);
        digit.write(2,(int)d / 10 % 10);
        digit.write(3,(int)d / 1 % 10);
        wait(1);
    }
}
```

読者プレゼントのお知らせ

『Software Design』をご愛読いただきありがとうございます。本誌サイト**http://sd.gihyo.jp/**の「読者アンケートと資料請求」にアクセスし、アンケートにご協力ください（アンケートに回答するにはgihyo.jpへのお名前と住所のアカウント登録が必要となります）。ご希望のプレゼント番号を記入いただいた方の中から抽選でプレゼントを差し上げます。締め切りは**2017年10月17日**です。プレゼントの発送まで日数がかかる場合がありますが、ご容赦ください。

ご記入いただいた個人情報は、プレゼントの抽選および発送以外の目的で使用することはありません。アンケートの回答は誌面作りのために集計いたしますが、それ以外の目的ではいっさい使用いたしません。記入いただいた個人情報は、作業終了後に責任を持って破棄いたします。

01 ネットワークストレージサーバ「F2-220」 1名



1,000ユーザ以上からの同時アクセスにも対応できる、ネットワークストレージサーバです。CPUは「Intel Celeron 2.41GHz デュアルコア」、メモリは「2GB RAM」。3.5/2.5インチSATA HDD、2.5インチSSDなどに対応し、最大容量は20TB（10TB×2ベイ）です（※HDDは付属しません）。ネットワークアダプタは1000BASE-T×1で、冗長構成としては、RAID 0/1、JBODおよびSingle Disk（シングルモード）をサポート。

提供元 TerraMaster, Inc. http://www.terra-master.com/html/jp

02 薄型モバイルバッテリー 世界のトランプ 2名



充電ケーブルを本体に収納できる、薄型のモバイルバッテリーです。容量は10,000mAhで、最大2.1A出力に対応。iPhone用（Linghtningケーブル）、Android用（MicroUSB端子ケーブル）のどちらか1つをプレゼント（画像はiPhone用）。

提供元 フォースメディア https://www.forcemedia.co.jp

03 ねこ手帳 2018 2名



ねこの写真が満載のスケジュール帳です。日記・家計簿としても使えます。ねこの生態やねこを上手に撮る方法などの豆知識も収録され、ねこ好きの方必携の手帳となっています。

提供元 技術評論社 http://gihyo.jp

04 TensorFlow機械学習クックブック 2名

Nick McClure 著、株式会社クイープ 訳



Googleが開発した機械学習ライブラリ「TensorFlow」について、基本からオープンデータを扱う方法、線形回帰やCNN/RNNのための具体的なレシピ、連立常微分方程式まで取り上げています。

提供元 インプレス https://www.impress.co.jp

05 暗号技術のすべて 2名

IPUSIRON 著

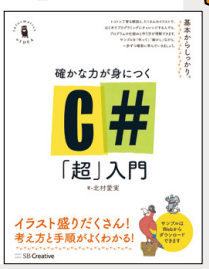


シーザー暗号といった古典暗号から、共通鍵暗号、公開鍵暗号といった現代暗号、ブロックチェーンなどの暗号技術を活用したセキュリティ技術まで、数式なども使ってひとつひとつ解説しています。

提供元 翔泳社 http://www.shoeisha.co.jp

06 確かな力が身につくC#「超」入門

北村 愛実 著



小さなプログラムを「Visual Studio Community」で動かしながら、C#の文法やVisual C#について学べます。オブジェクト指向入門者用の解説も扱われています。

提供元 SBクリエイティブ http://www.sbcr.jp

07 Mackerelサーバ監視［実践］入門 2名

井上 大輔、ほか 著



はてなの開発したSaaS型サーバ監視ツール「Mackerel」の入門書です。Mackerelの使い方だけではなく、はてなのエンジニアが培ってきたインフラ管理のノウハウについても学べる1冊になっています。

提供元 技術評論社 http://gihyo.jp

第1特集

個人でも会社でも使える！

これだけは知っておきたい Gitのキホン

GitHub Bitbucket GitLab GitBucket

Gitを使いこなしてますか？　まだでしたらこの特集でGitの操作を覚えてしまいましょう。まずは1人で、ソースコードのバックアップをとるイメージで管理して、操作に慣れてください。

プログラマにとっては毎日の作業ですので、操作に慣れたらできるだけラクをしたくなるでしょう。コードを書いているエディタから離れずにGit操作ができたら便利です。本特集で紹介するVimとAtomで、そのスマートさを感じてください。

GitHub、Bitbucket、GitLab、GitBucketの違いを知り、開発現場での利用方法を参考に、より良いソースコード管理をしましょう。

第1章 GitHub、Bitbucket、GitLab、GitBucket徹底比較！
Gitサービス／プロダクトの選び方 18
AUTHOR 竹添 直樹

第2章 Gitの基礎から実践までを学ぼう
インストールから複数人での共同作業運用までのGit操作 26
AUTHOR 豊沢 泰尚、久保 靖資

第3章 構造を知るとGitの真価がよくわかる
Gitの内部構造とコマンドの作用の背景 39
AUTHOR 豊沢 泰尚、久保 靖資

第4章 強力プラグインgina.vim
実践！ エディタからのGit操作［Vim編］ 46
AUTHOR Λlisue

第5章 GitからGitHubまでエディタから操作して作業がはかどる！
実践！ エディタからのGit操作［Atom編］ 53
AUTHOR 大竹 智也

第6章 Git活用の実例［1］
クラウド＝GitHubとオンプレ＝GitLabを使い分けるピクシブ 60
AUTHOR 山際 康貴

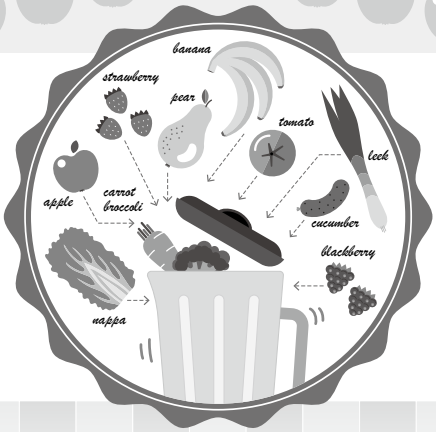
第7章 Git活用の実例［2］
GitLab CI＋RedPenでドキュメント検査もCIしよう！ 64
AUTHOR 打田 智子



第1章

GitHub、Bitbucket、GitLab、GitBucket徹底比較！

Gitサービス／プロダクトの選び方



Gitでのソースコード管理と聞くと、まずGitHubが連想されるのではないでしょうか。ですが、Gitによるバージョン管理システムを提供するサービス／プロダクトはGitHub以外にもたくさんあります。特集の導入にあたる本章では、メジャーな4つのサービス／プロダクトの特徴を解説します。

AUTHOR 竹添 直樹（たけぞえ なおき）
㈱ビズリーチ
TWITTER @takezoen



さまざまなGitサービス／プロダクト、どれを使う?

ここ数年、ソースコードのバージョン管理システム[注1]はGitが主流になっています。単にGitを使うだけであればローカルディスク上にGitリポジトリを作成して利用することもできますが、複数人でソースコードを共有したい場合はGitサーバが必要になります。Gitサーバを立てる方法はおおまかに次のどちらかになります。

・インターネット上で提供されているホスティングサービス（SaaS）を使う
・オンプレミスやクラウドなど任意の環境に設置可能なパッケージを使う

前者であればGitHub、Bitbucketなど、後者であればGitHubのパッケージ版であるGitHub Enterprise、同じくBitbucketのパッケージ版であるBitbucket Server、さらにはオープンソースのGitLabやGitBucketなどさまざまな選択肢が存在します。また、SaaSの一種と言えますが、Amazon Web Services（AWS）などのクラウドサービスを利用している場合はそれらのクラウドサービスで提供されているGitリポジトリのホスティング機能を使用するという選択肢もあります。**表1**に主なサービス／プロダクトをまとめました。

これらのサービス／プロダクトの中からどれを使うべきなのでしょうか。まずはそれぞれの方法についてメリット・デメリットをおおまかに見ていきましょう。



インターネット上で提供されているホスティングサービスを使う

自前でサーバをセットアップしたりする必要もなく、運用の手間もないので最も手軽な方法です。多くのサービスはパブリック（公開）リポジトリであれば無料で作成することができます。プライベート（非公開）リポジトリは有料プランでないと作れなかったり、無料プランだと制限があったりするものの、自前でGitサーバを構築・運用する手間や費用を考えれば安いものです。筆者としては特別な理由がない限りはホスティングサービスの利用をお勧めします。



オンプレミスやクラウドなど任意の環境に設置可能なパッケージを使う

業務での開発ではセキュリティ要件などによりソースコードの管理に外部のサービスを利用できないケースもあります。このような場合は自前でGitサーバを構築する必要があるわけで

注1）バージョン管理システムはソースコードの過去のすべての変更履歴を記録でき、過去にどのような変更が行われたかをあとから参照したり、任意の時点のソースコードに戻したりといったことが可能です。また、ソースコードを複数人で共有することもできます。

▼表1　Gitを利用可能なサービス／プロダクト

サービス／プロダクト名	URL	種別	有料／無料	備考
GitHub	https://github.com/	SaaS	無料(有料プランあり)	パッケージ版のGitHub Enterprise(有料)もあり
Bitbucket	https://bitbucket.org/	SaaS	無料(有料プランあり)	Git、Mercurialをサポート。パッケージ版のBitbucket Server(有料)もあり
GitLab	https://about.gitlab.com/	パッケージ	無料(有料版あり)	無料版はオープンソース。SaaS(無料／有料)もあり
GitBucket	https://gitbucket.github.io/	パッケージ	無料	オープンソース
Gogs	https://gogs.io/	パッケージ	無料	オープンソース
OSDN	https://ja.osdn.net/	SaaS	無料	Git、Mercurial、Subversionをサポート
SourceForge.net	https://sourceforge.net/	SaaS	無料	Git、Mercurial、Subversionをサポート
Apache Allura	https://allura.apache.org/	パッケージ	無料	オープンソース(SourceForge.netの基盤がOSS化されたもの)
Gitblit	http://gitblit.com/	パッケージ	無料	オープンソース
Backlog	http://www.backlog.jp/	SaaS	有料	プロジェクト管理ツールの機能としてGit、Subversionをサポート
AWS CodeCommit	https://aws.amazon.com/jp/codecommit/	クラウド	有料	Amazon SNSやAWS Lambdaとの連携が可能
Cloud Source Repositories	https://cloud.google.com/source-repositories/	クラウド	有料	Stackdriver Debuggerと連携可能

すが、GitHubやBitbucketはパッケージ版を有料で提供していますし、GitLabやGitBucketといったオープンソースソフトウェアもあり、選択肢は豊富です。ただし、有料製品はオンラインのホスティングサービスと比べると割高な傾向がありますし、オープンソースのものを使ったとしてもサーバの調達や運用コストはばかになりません。



クラウドサービスで提供されているホスティング機能を使う

AWS、Google Cloud Platform（GCP）などのクラウドサービスで提供されているGitリポジトリのホスティング機能は、おもにそれぞれのクラウドサービスとの連携を目的としたものになっており、リポジトリのホスティング以外の機能（課題管理やプルリクエスト、Wikiなど）は基本的に備えていません。本稿では詳細には取り上げませんが、Gitリポジトリだけあればよいという場合や、クラウドサービスとの連携機能に価値を見いだせるのであれば利用を検討してもよいでしょう。

◆　◆　◆

以降では主な選択肢として次の4つのサービス／パッケージについて詳しく紹介していきます。**表2**にこれらの料金プランをまとめました。

- GitHub
- Bitbucket
- GitLab
- GitBucket



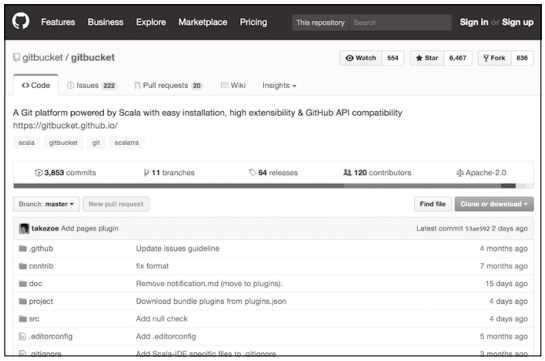
圧倒的なデファクトスタンダード「GitHub」

GitHubは米GitHub社が提供する、ソフトウェア開発を総合的に支援するサービスです（**図1**）。Gitリポジトリのホスティングだけでなく、バグやタスクを管理するための機能や、Wikiによるドキュメント管理機能なども提供されてい

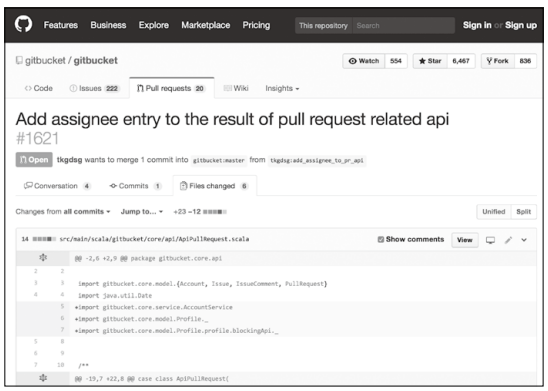
▼表2　GitHub、Bitbucket、GitLab、GitBucketの料金プラン

サービス／プロダクト名	プラン	料金	プライベートリポジトリ	CI	24時間サポート	特記事項
GitHub（SaaS）	Free	無料	なし	外部サービスと連携	なし	―
	Developer（個人アカウント）	1ユーザあたり7ドル/月	無制限	外部サービスと連携	なし	―
	Team（組織アカウント）	1ユーザあたり9ドル/月	無制限	外部サービスと連携	なし	―
	Business（組織アカウント）	1ユーザあたり21ドル/月	無制限	外部サービスと連携	営業日のみ	シングルサインオンなどを利用可能
GitHub Enterprise		10ユーザあたり2,500ドル/年	無制限	外部サービスと連携	365日	―
Bitbucket（SaaS）	Free	無料	無制限（共有できるのは5ユーザまで）	Bitbucket Pipelines（50分/月まで）	別契約が必要	JIRAと連携可能。1GB/月のLFSストレージ。CIとLFSは追加料金で拡張可能
	Standard	1ユーザあたり2ドル/月（最小10ドル/月から）	無制限	Bitbucket Pipelines（500分/月まで）	別契約が必要	JIRAと連携可能。5GB/月のLFSストレージ。CIとLFSは追加料金で拡張可能
	Premium	1ユーザあたり5ドル/月（最小25ドル/月から）	無制限	Bitbucket Pipelines（1,000分/月まで）	別契約が必要	JIRAと連携可能。10GB/月のLFSストレージ。CIとLFSは追加料金で拡張可能。管理者向け・セキュリティ関連の機能が追加される
Bitbucket Server	10ユーザまで	買い切り10ドル	無制限	Bambooと連携可能	別契約が必要	Bamboo、JIRA、Confluenceと連携可能
	25ユーザ以上	買い切り1,800ドル〜	無制限	Bambooと連携可能	別契約が必要	Bamboo、JIRA、Confluenceと連携可能
GitLab.com（SaaS）	Free	無料	無制限	GitLab CI（2,000分/月まで）	なし	―
	Bronze	1ユーザあたり4ドル/月	無制限	GitLab CI（2,000分/月まで）	なし	タスクボードやマージリクエストを中心にさまざまな機能が追加される
	Silver	1ユーザあたり19ドル/月	無制限	GitLab CI（10,000分/月まで）	365日	機能的にはBronzeとほぼ同じだが、サポートなどが追加される
	Gold	1ユーザあたり99ドル/月	無制限	GitLab CI（50,000分/月まで）	365日	CIのキャパシティが増える以外はSilverと同じ
GitLab	Community Edition	無料	無制限	GitLab CI	なし	―
	Enterprise Edition	1ユーザあたり3.13ドル/月	無制限	GitLab CI	365日	Bronze相当の機能に加えてLDAP連携などが可能
	Enterprise Edition Premium	1ユーザあたり16.59ドル/月	無制限	GitLab CI	365日	Silver相当の機能、サポートを利用可能
GitBucket		無料	無制限	Jenkinsと連携可能	なし	プラグインで機能を拡張可能

▼図1　GitHub



▼図2　プルリクエスト



ます。多くのオープンソースソフトウェア（OSS）の開発に利用されており、Gitのホスティングサービスとしてはデファクトスタンダード的な存在になっています。

GitHubは基本的に無料で使用することができますが、パブリックリポジトリしか作成することができません。プライベートリポジトリを作りたい場合は有料プランを契約する必要があります。

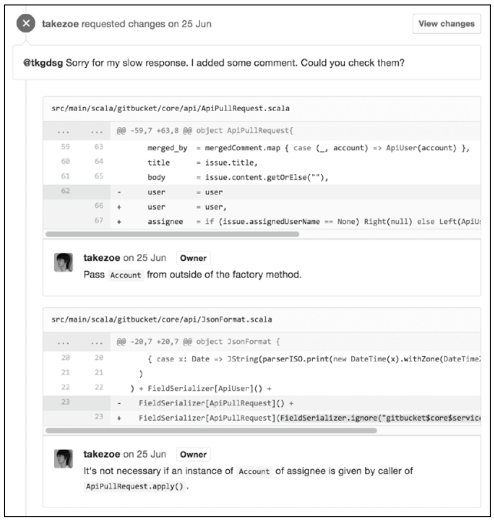


GitHubの人気の秘訣

プルリクエストによる開発スタイル

同種のオンラインサービスとしてはGitHub以前にもSourceForge.net、Google Code[注2]などが存在し、人気を集めていましたが、GitHubでは「プルリクエスト」という第三者が

▼図3　差分を確認しながらコードレビューができる



気軽に修正をフィードバックできる機能があり、これがOSSの開発スタイルに非常にマッチしていました（**図2**）。

それまでのOSS開発では、外部のプログラマがソフトウェアに対してバグ修正や機能追加を行う際にはソースコードの差分をパッチとしてメーリングリストやバグトラッカーなどに投稿し、レビューを経たうえで取り込まれるというスタイルが一般的で、非常に面倒かつハードルの高いものでした。

これに対しGitHubでは誰でも簡単にリポジトリをフォーク（複製）でき、フォークしたリポジトリに対して行った変更を、オリジナルのリポジトリに対してプルリクエストという形で送信することができたのです。さらにプルリクエストに対してWebブラウザ上でソースコードの差分を確認しながらコメントを入れることができ（**図3**）、非常に手軽かつわかりやすい形でレビューを行うことが可能です。

このようにソフトウェア開発に直接的に必要となる機能だけでなく、ユーザからのフィードバックやレビューといったコミュニケーションを促進する機能を提供していた点が、GitHubと従来のサービスとの決定的な違いであり、GitHubが「ソーシャルコーディング」を標榜（ひょうぼう）し

注2）現在はサービス終了。

ているゆえんでもあります。

GitHubを中心としたエコシステム

ほかのサービスと比べて機能面でGitHubが劣っている部分もあります。たとえばGitHubにはProjectsというシンプルなタスクボード機能がありますが、後述するGitLabはイシューのステータスと連動し、タイムトラッキングも可能な、より高機能なタスクボードが提供されています。また、BitbucketやGitLabはCI[注3]機能が統合されていますが、GitHubには機能そのものが存在しません。

しかし、GitHubにはCIサービスのTravis CI（**図4**）、タスクボードやプロジェクト管理機能を提供するZenHub（**図5**）、チャットサービスのGitter（**図6**）など、サードパーティ製のツールや連携可能な外部サービスが数多く存在し、これらを利用することで不足している機能をカバーすることができます。また、Heroku[注4]やDocker Hub[注5]のようにGitHubからデプロイを行うサービスもあります。GitHubを中心としたエコシステムに乗ることで、これらのツールやサービスの恩恵も受けられるという点もGitHubを利用する大きなメリットといえます。



GitHubを業務で利用する場合

業務での開発ではソースコードを外部に公開できないケースが多いのではないでしょうか。

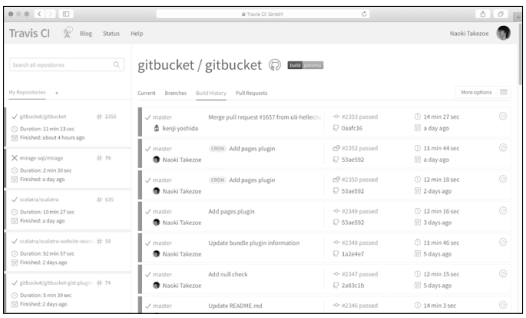
GitHubでは有料プランを利用することでプライベートリポジトリを無制限に作成することができます。また、上位プランであるBusinessプランでは、シングルサインオンなどエンタープライズ向けの機能や24時間サポートも利用可能です。

セキュリティ要件などの理由からオンラインのホスティングサービスを利用できない場合はGitHub Enterpriseが利用できます。GitHub Enterpriseは仮想マシンイメージとして提供され、オンプレミス環境だけでなくAWSやGCP、Azureなどのクラウドサービスにも設置することができます。

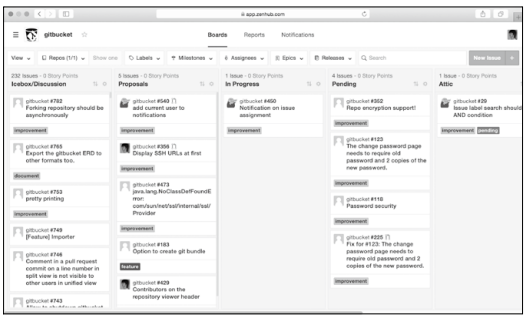
注3）Continuous Integration（継続的インテグレーション）：ソフトウェアの品質保証のためにビルドやテストを継続的に行う手法のこと。

注4）URL https://www.heroku.com/
注5）URL https://hub.docker.com/

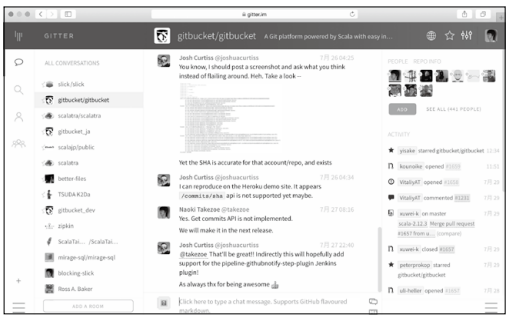
▼図4　Travis CI（https://travis-ci.org/）



▼図5　ZenHub（https://www.zenhub.com/）



▼図6　Gitter（https://gitter.im/）





コストパフォーマンスなら「Bitbucket」

Bitbucketは豪Atlassian（アトラシアン）社が提供するサービスです（**図7**）。当初は独立したスタートアップが運営するサービスでしたが、2010年にアトラシアンによって買収されました。また当初はMercurialというバージョン管理システムのみサポートしていましたが、アトラシアンによる買収後の2011年にGitのサポートが追加されました。



無料でもプライベートリポジトリを作成可能

Bitbucketはユーザインターフェースこそ違えど、現在ではGitHubとほぼ同等の機能を備えています。GitHubとの最大の違いは、共同作業可能な人数に5人までという制限はあるものの「無料でもプライベートリポジトリを作成できる」という点です。そのため、数名でのちょっとした共同作業にプライベートリポジトリを使いたいといった用途であればBitbucketが最適です。有料プランではプライベートリポジトリを無制限に共有できるようになりますが、こちらの価格もGitHubと比べると割安です。



CIツールの統合

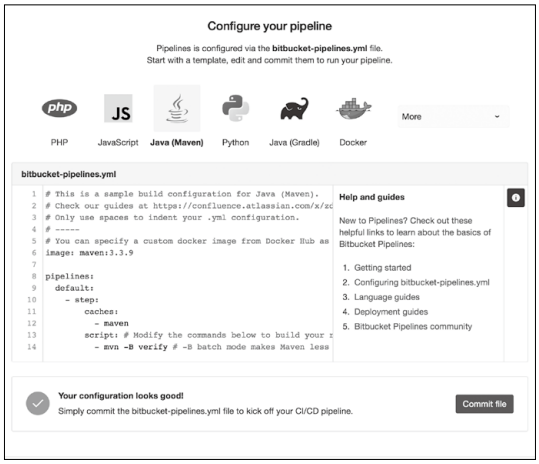
Bitbucketのもう1つの特徴が、Bitbucket PipelinesというCI機能が統合されているという点です（パッケージ版のBitbucket Serverではこの機能は利用できず、Bambooというアトラシアンが提供するCI製品との連携が推奨されています）。前述したようにGitHubもTravis CIやCircle CIなどの外部サービスやJenkinsなどとの連携は可能ですが、Bitbucketではデフォルトで CI機能が統合されており、Web画面から対象のプログラミング言語を選択するだけで設定のひな形を作成できます（**図8**、**図9**）。



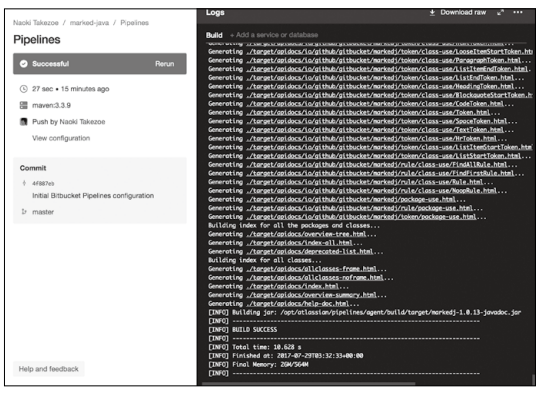
パッケージ型のBitbucket Serverも

GitHubにGitHub Enterpriseがあるように、Bitbucketにもパッケージ版のBitbucket Serverという製品が存在します。ユーザ数に応じた買い切りモデルになっているので月々の

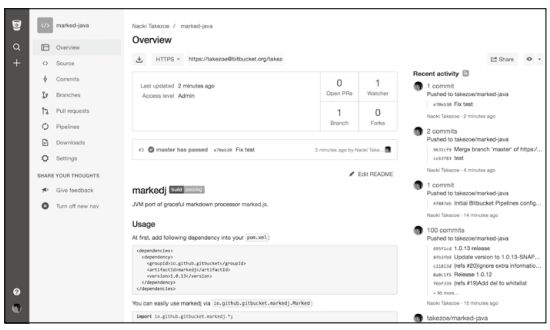
▼図8　言語を選択するだけでCI設定のひな形を作成できる



▼図9　CIの実行結果



▼図7　Bitbucket



利用料が発生しません。また、少人数向けのプランが用意されているため小さなチームでも導入しやすくなっています。セキュリティや監査対応などの機能のために商用製品を利用したいものの、コストはできるだけ抑えたいという場合にはBitbucket Serverが有力な候補になるでしょう。ただし、24時間サポートについては有料プランでも別途サポート契約が必要になるので注意が必要です。

また、Bitbucket Serverは同じくアトラシアンが提供するJIRA（課題管理システム）、Bamboo（CIツール）、Confluence（ドキュメント管理システム）との連携が可能です。すでにこれらのツールを使用している場合や、あわせて導入を検討している場合はその親和性や、サポートコストをまとめることができるという点もBitbucket Serverのアドバンテージになるかもしれません。



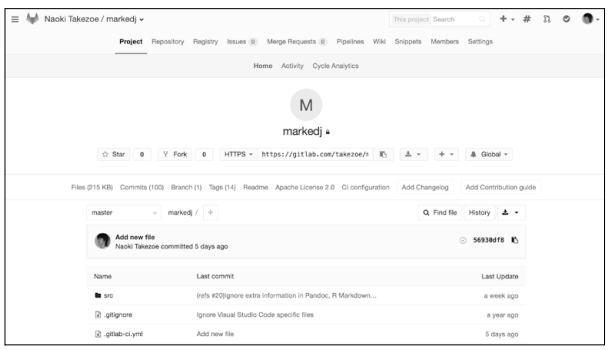
オープンソースという選択肢「GitLab」

GitLabは、米GitLab社が開発するオープンソースソフトウェアで、おもにRuby（一部Go言語）で作成されています（**図10**）。もともとは完全なオープンソースソフトウェアとして作成されたものでしたが、2013年にオープンソースのCommunity Editionと有料のEnterprise Editionに分割され、現在に至ります。歴史的経緯からインストール型のGitサーバとして認識されていることが多いのですが、GitLab.comというSaaS版も提供しています。

機能的にはGitHubでできることは一通り可能ですし、Bitbucket同様CI機能も統合されています。また、標準でDockerコンテナレジストリやタイムトラッキングも可能なタスクボードも利用可能です。さらにSaaS／パッケージ版ともに有料プランではバーンダウンチャートなどの機能やサポートが利用可能なほか、SaaS版では有料プランを選択することでCIのキャパシティも追加されます。一方で無料プランではイシューを複数人にアサインできないなど、GitHubでは無料プランでもサポートされている機能が利用できなかったりします。

GitLabは企業が販売する製品でありながらオープンソースでの開発も活発に行われていますので、何か問題があった場合や追加したい機能がある場合には自分でコントリビュートすることも可能です。オープンソースであるという点に価値を見いだせるのであれば、GitLabを選択するメリットは大きいのではないかと思います。

▼図10　GitLab





設置が簡単、プラグインで拡張可能な「GitBucket」

GitBucketはGitLabと同じくオープンソースで開発されているGitサーバです（**図11**）。こちらはScalaで開発されており、Java（Java 8以降が必要です）さえインストールされていれば動作させることができるのでWindows上での利用も容易です。ここまでに紹介してきたGitHub/Bitbucket/GitLabと異なり、オンラインでのホスティングサービスは提供されておらず、有料バージョンなどもありません。機能としてはGitリポジトリのホスティング、イシューやプルリクエスト、WikiなどGitHubでできることは一通りできるようになっています。

GitBucketの特徴はなんといってもその導入の容易さです。公式サイトからたどれるダウンロードページからgitbucket.warというファイ

ルをダウンロードして、次のコマンドを実行するだけで起動できます。

```
$ java -jar gitbucket.war
```

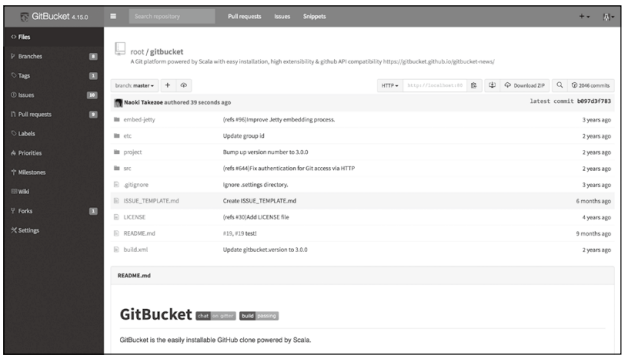
デフォルトではH2という組み込みデータベースを使用しますが、本格的な用途に利用する場合は性能面やデータの安全性の面から外部データベース（MySQLとPostgreSQLに対応しています）の利用が推奨されています。

また、プラグインによる拡張も可能となっており、GitBucket Community Plugins[注6]というサイトに有志の手によって作成されたプラグインがカタログ化されています（**表3**）。もちろん自分でプラグインを作成することもできますので、利用シーンに応じたカスタマイズが可能です。

GitBucketは本稿で紹介しているほかのプロダクトと異なり、企業ベースではなくボランタリベースで開発が行われているため開発速度や機能の充実度で劣る面があるのは否めません。しかし手軽にインストールすることができるので、ちょっとした用途にGitサーバが必要な場合などに手軽に利用することができます。また、プラグインによる拡張の自由度があるので必要な機能を自分で追加していくこともできます。

◆ ◆ ◆

▼図11　GitBucket



Gitを利用可能なサービス／プロダクトにはさまざまな選択肢がありますが、GitHubが圧倒的なデファクトスタンダードに君臨しています。少なくともソースコードを公開可能な用途であればGitHub以外を選択するメリットは考えにくいのではないでしょうか。ソースコードを公開できない場合や、業務で利用する場合は予算や要件に応じてGitHub以外の選択肢も検討するといいでしょう。

ソースコードのバージョン管理システムはソフトウェア開発の要とも言える重要なものです。用途にあわせて適切なものを選択するようにしましょう。SD

注6）URL https://gitbucket-plugins.github.io/

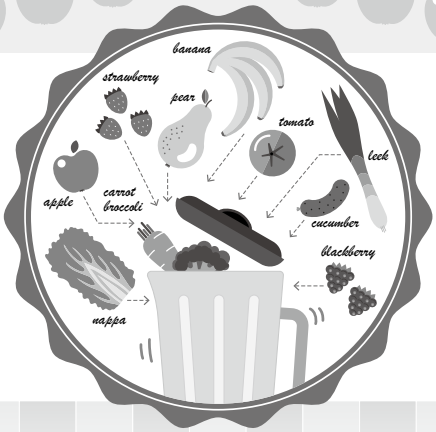
▼表3　GitBucketの主なプラグイン

プラグイン名	説明
Gist plugin	スニペットをGitリポジトリで管理できるようにする
AsciiDoc plugin	MarkdownだけでなくAsciiDocをレンダリングできるようにする
Pages plugin	リポジトリの内容をWebサイトとして公開できるようにする
Network plugin	リポジトリのコミット履歴をビジュアルに表示する
Emoji plugin	絵文字を使えるようにする
RST plugin	MarkdownだけでなくReSTructured Textをレンダリングできるようにする
Explorer plugin	リポジトリにツリー状のナビゲーションを追加する
FESS plugin	全文検索サーバFessを使って全文検索できるようにする
Jupyter plugin	Jupyterのノートブックをレンダリングできるようにする
Notification plugin	イシューやプルリクエストの変更をメールで通知する



Gitの基礎から実践までを学ぼう

第2章 インストールから複数人での共同作業運用までのGit操作



本章では、Gitのインストールから初期設定の方法を説明します。また、インターネットを介してBitbucketを利用し、リモートリポジトリの作成からブランチの操作を行います。そして最後に、1人で作業する場合と、複数人で共同作業を行う一連の動作を学びます。

AUTHOR 豊沢 泰尚（とよさわ やすたか）
㈲シェアリングシード
AUTHOR 久保 靖資（くぼ やすし）
㈱エクスパ

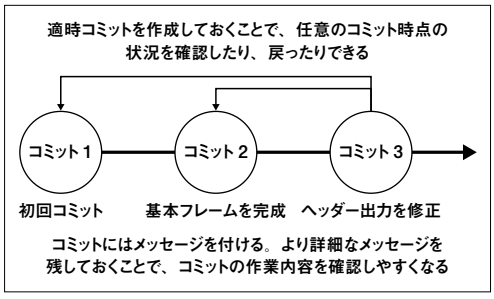


Gitによるバージョン管理運用の概要

Gitによるバージョン管理は、「コミット」と呼ばれる作業履歴をリポジトリに対して記録することで行われます。コミットは時系列に沿って記録され、過去に記録したコミットに戻ることで、Gitで管理するプロジェクトデータを過去の状態に戻すことができます。

したがって、コミット単位で「それがどのような作業内容だったのか」を明示的に記しておくことが重要です。そのためコミットを記録する際には必ず「コミットメッセージ」をつける必要があります。Gitによるバージョン管理作業は、基本的に「作成・修正などの作業」→「コミット」の繰り返しです（**図1**）。

▼図1 Gitの基本運用は、プロジェクトを管理するリポジトリに対して適時「作業内容をコミット」すること





Gitのインストールと初期設定

Gitはさまざまなosで利用できますが、ここではWindowsとCentOSでのインストールと初期設定について紹介します。



Windowsでのインストール

WindowsにGitをインストールする場合、git for windowsのWebサイト[注1]にアクセスしてインストーラをダウンロードして実行するだけです。



CentOSでのインストール

CentOSにGitをインストールする場合は、yumコマンドを利用し、

```
$ sudo yum install -y git
```

とするだけです。



Gitの初期設定

インストールしたばかりのGitでは、いくつか初期設定を行う必要があります。まずはコミットに記録される名前とメールアドレスを設定しましょう。コマンドプロンプトやターミナルな

注1） URL https://git-for-windows.github.io/

どで、次のGitコマンドを実行します。

```
$ git config --global user.name "MyName"
$ git config --global user.email ↵
"myemail@example.com"
```

1行目が名前の設定、2行目がメールアドレスの設定となります。



Bitbucketを利用した1人作業

まず、共同作業ではなく、1人で開発作業をしている場合のGitによるバージョン管理を紹介しましょう。インターネットを介してアクセスできるリモートリポジトリをバックアップ先として、ローカルリポジトリでの作業をコミット+プッシュによりバックアップしておく、というフローになります。

この作業のメリットは、作業内容を適時記録(コミット)すると、それをローカルPCだけでなくネット上のリモートリポジトリにバックアップできるという点です。適時コミットを記録しておけば、過去にどのような作業を行ったかを時系列に沿って確認できるようになります。



Bitbucketにリモートリポジトリを作成する

ここではBitbucket[注2]というオンラインのGitサービスを利用して、リモートリポジトリを作成します。Bitbucketはアトラシアン社が運営しているホスティングサービスで、5ユーザまでであれば無料でGitリポジトリを作成でき、非公開であればリポジトリ数は無制限です。1人作業、2〜5名程度の小規模な開発体制には最適なサービスといえます。

利用にあたっては会員登録が必要です。ブラウザで「https://ja.atlassian.com/software/bitbucket」にアクセスし、[無料で始める]ボタンをクリックし、アカウント(メールアドレス)を登録しましょう。アカウント登録が完了したら、ログイン画面にアクセスして、メールアドレスとパスワードでマイページにログイン。ダッシュボードが表示されます。

リモートリポジトリを作成し、URLを取得

では、さっそくBitbucketにリモートリポジトリを作成しましょう。ダッシュボード画面の左側にある「+」ボタン(Create)をクリックして表示される左ペイン画面の「Repository」をクリックします。すると新規リポジトリの作成画面が表示されます。リポジトリ名に任意の名前を入力します。そしてアクセスレベル「これは非公開リポジトリです」をチェックし、リポジトリタイプは「Git」を選んで[リポジトリの作成]ボタンをクリックします。これで新規リポジトリ作成は完了です。

ここで作成したリポジトリには、まだ作業データは登録されていません。つまり、これから作業データを入れるための「箱」を用意しただけといえます。

新規リポジトリを作成すると、そのリポジトリの概要画面が表示されます。この画面の右上部にリポジトリにアクセスするためのURLが表示されています(**図2**)。次にローカルPCでリモートリポジトリをCloneする必要があるため、このURLをコピーするなど、記憶しておきます。



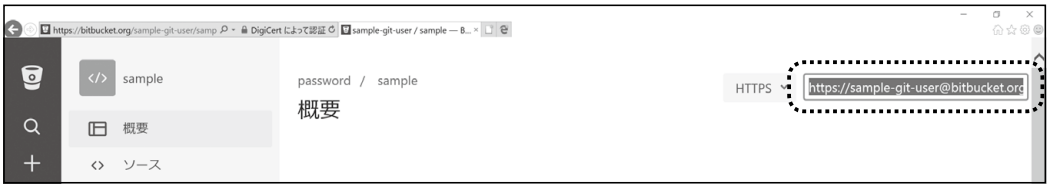
Gitリポジトリを GUI操作する

Bitbucketを運用するアトラシアン社では、ローカルPCでGitリポジトリをGUI操作できる「SourceTree」[注A]というアプリケーションも無料で提供しています。SourceTreeを使えば、コマンド入力ではなく、マウス操作でGitを利用できるため、初心者でも簡単にGitバージョン管理を行えます。

注A) URL https://ja.atlassian.com/software/sourcetree

注2) URL https://bitbucket.org/

▼図2　作成したリモートリポジトリをCloneするために必要なURL情報を確認



Bitbucketのリポジトリを手元のPCに複製（git clone）

Bitbucket上にリポジトリを作成したら、次にローカルPCにそのリポジトリの複製（クローン）を作成しましょう。リモートリポジトリを複製するには、リモートリポジトリのURLを利用して、git cloneというコマンドを実行します（**図3**）。このコマンドを実行すると、ローカルPCに./sampleディレクトリが作成され、その中にGitリポジトリが作成されます。

この時点では、リモートリポジトリには何のデータも登録されていないので、ローカルPCに複製したローカルリポジトリも空の状態です。

現在の作業の状況を確認しよう（git status）

ローカルPCに作成された./sampleディレクトリに開発作業用のプログラムファイルなどを設置します。例として、次のように3つのPHPファイルを設置したとします。ディレクトリの内容は次のとおりです。

```
./sample
├── builder.php
├── index.php
└── show.php
```

ここで、現在の状況を確認してみましょう。git statusコマンドを実行します（**図4**）。

この結果を見ると3つのPHPファイルが存在しますが、いずれも「Untracked files」として表示されています。これは、まだどのファイルもGitの管理下に含まれていないことを示しています。

コミットしてスナップショットを記録しよう（git add、git commit）

配置した3つのPHPファイルをリポジトリに登録してみましょう。Gitリポジトリにファイルを登録するには、まずは登録したいファイルをステージにセットし、その後コミットする、というフローになります。そこで、まずは次のコマンドを実行します。

▼図3　ローカルPCにリポジトリのクローンを作成

```
$ git clone https://sample-git-user@bitbucket.org/sample-git-user/sample.git
Cloning into 'sample'...
Password for 'https://sample-git-user@bitbucket.org':
warning: You appear to have cloned an empty repository.
Checking connectivity... done.
```

▼図4　現在の状況を確認する

```
$ git status
On branch master
Initial commit
Untracked files:
(use "git add <file>..." to include in what will be committed)
builder.php
index.php
show.php
nothing added to commit but untracked files present (use "git add" to track)
```

```
$ git add -A
```

コマンドの実行結果はとくに何も表示されません。git addコマンドはファイルをステージにセットするもので、「-A」オプションはすべてのファイルをステージにセットするという意味になります。個々にファイル名を指定することもできます。

ここで、git statusコマンドで再度作業状況を確認してみます（**図5**）。

git addコマンドによってファイルをトラッキングしたあとでは、上記コマンド実行結果のように「Changes to be committed」、つまりコミットされる予定のファイルとしてステージにセットされているのがわかります。

この状態でコミットすることで、3つのファイルをリポジトリに登録できます。**図6**のように、git commitコマンドを実行しましょう。

先にも述べましたが、コミットする際はメッセージを同時に記録することが必須です。コミットメッセージは「-m」オプションによって指定します。今回は最初のコミットとなるため「initialize repositry」としましたが、「first commit」など作業内容が判別できるのであれば任意のメッセージでかまいません。

これでコミット作業が完了し、「initialize repositry」というコミットが作成され、3つのファイルが登録されたことになります。

▼図5　再度、状況を確認する

```
$ git status
   On branch master

   Initial commit

   Changes to be committed:
     (use "git rm --cached <file>..." to unstage)

           new file:   builder.php
           new file:   index.php
           new file:   show.php
```

▼図6　コミットして3つのファイルをリポジトリに登録する

```
$ git commit -m"initialize repositry."
[master (root-commit) d2172ab] initialize repositry.
3 files changed, 180 insertions(+)
create mode 100644 builder.php
create mode 100644 index.php
create mode 100644 show.php
```

ファイルを編集して、編集前の状態との差分を確認（git diff）

初回コミットが完了したあと、ローカルPC上で作業したファイルを適時コミットしてバージョン履歴を残していく、というのがGitの基本的なワークフローとなります。

まずは、初回コミット後に行った変更個所の確認を行ってみましょう。git diffコマンドを使います（**図7**）。

git diffコマンドを実行すると、チェックアウトしている最終コミットに登録されているファイルと現在のファイルが比較され、相違のあるファイルとその差分個所が表示されます。

図7の例ではbuilder.phpというファイルが編集されており、その差分として文頭に「-」が付いた行で修正前、「+」が付いた行で修正後の内容が表示されています。このようにGitを利用すれば、前回のコミット以降に行った作業内容を簡単に確認できます。

続けて、今回の修正作業内容をコミットしてリポジトリに登録してみましょう。git addでステージセットし、git commitコマンドを実行、という流れです（**図8**）。今回は「Bug Fix #1」というコミットメッセージで登録します。

実行結果としては、差分ファイルのみがコミットされたことがわかります。このように、Gitでは前回のコミットとの差分を

▼図7　git diffコマンドによる変更個所の確認

```
$ git diff
diff --git a/builder.php b/builder.php
index 1fd19d4..af3e343 100644
--- a/builder.php
+++ b/builder.php
@@ -113,7 +113,7 @@ class SQLBuilder
// -- Primitiveの表現
public function stExpression($k)
{
- if (preg_match('!^[\w\d_\.]+$!',$k)) return $this->stName($k);  ←修正前
+ if (preg_match('!^[\w\d_]+(\.[\w\d_]+)*$!',$k)) return $this->stName($k);  ←修正後
return $k;
}
public function stName($k)
```

自動認識・抽出し、それを新しいコミットとして簡単に登録できます。

コミットされた履歴を確認しよう（git log）

Gitリポジトリに登録したコミットの履歴は、git logコマンドで確認できます。まずは、コミット履歴を時系列で確認してみましょう（**図9**）。「--stat」オプションを利用します。

▼図8　git commit -mコマンドで修正作業内容をコミットしてリポジトリに登録する

```
$ git add -A
$ git commit -m"Bug Fix #1"
[master d08954a] Bug Fix #1
1 file changed, 1 insertion(+), 1 deletion(-)
```

実行結果では、「Bug Fix #1」と「initialize repositry.」という2つのコミットとその内容が確認できます。また「-p」オプションを利用すると、任意のファイルの修正履歴を確認できます（**図10**）。

「-p」オプションでbuilder.phpを指定した実行結果では、該当ファイルに加えられた修正履歴が確認できます。先ほど、修正後のコミット作業でも確認できた行単位での修正前後の内容が履歴として保存されていることがわかります。このようにGitを利用すれば、修正以前の状態を失うことなく、作業内容のバージョン管理を行えることがわかります。

▼図9　git logでコミット履歴を表示する

```
$ git log --stat
commit d08954a7495fc078fe2f828eefd3652adcbfa02d
Author: DevelopmentTeam <dev@localhost>
Date: Sat Jul 29 01:49:38 2017 +0900

Bug Fix #1  ←2つ目のコミット

builder.php | 2 +-
1 file changed, 1 insertion(+), 1 deletion(-)
commit d2172ab401d845c447aac298db51c0a2cc64e7d5
Author: DevelopmentTeam <dev@localhost>
Date: Sat Jul 29 01:39:00 2017 +0900

initialize repositry.  ←1つ目のコミット

builder.php | 151 ++++++
```

▼図10　任意のファイルの修正履歴を確認する

```
$ git log -p builder.php
commit d08954a7495fc078fe2f828eefd3652adcbfa02d
Author: DevelopmentTeam <dev@localhost>
Date: Sat Jul 29 01:49:38 2017 +0900

Bug Fix #1

diff --git a/builder.php b/builder.php
index 1fd19d4..af3e343 100644
--- a/builder.php
+++ b/builder.php
@@ -113,7 +113,7 @@ class SQLBuilder
// -- Primitiveの表現
public function stExpression($k)
{
- if (preg_match('!^[\w\d_\.]+$!',$k)) return $this->stName($k);
+ if (preg_match('!^[\w\d_]+(\.[\w\d_]+)*$!',$k)) return $this->stName($k);
return $k;
}
public function stName($k)
```

▼図11　リモートリポジトリにコミット履歴を送信する

```
$ git push origin master
Password for 'https://sample-git-user@bitbucket.org':  ←パスワード入力を要求される
Counting objects: 8, done.
Compressing objects: 100% (8/8), done.
Writing objects: 100% (8/8), 2.69 KiB | 0 bytes/s, done.
Total 8 (delta 1), reused 0 (delta 0)
To https://sample-git-user@bitbucket.org/sample-git-user/sample.git
* [new branch] master -> master
```

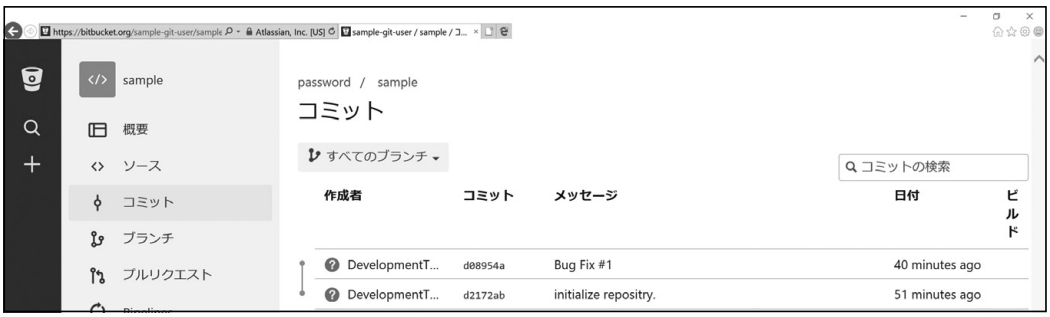
リモートリポジトリにローカルのコミット履歴を送信（git push）

ローカルでの作業が完了したら、リモートリポジトリにコミット履歴を送信してみましょう。これは、ローカルPCでの作業をバックアップするという意味にもなります。git pushコマンドを使います（**図11**）。

ここで「push」とは複製元のリモートリポジトリへアップロードするという意味になります。また「origin」はgit clone元となったリモートリポジトリのデフォルトのニックネーム（サーバ名）、「master」はデフォルトのブランチ名を意味します。

図11のように、まずはBitbucketに登録したアカウントのパスワード入力を求められます。パスワードを入力すると、あとはアップロード

▼図12　ブラウザでBitbucketのリモートリポジトリにアクセスしてブランチの状態を確認する



作業が自動処理されます。実行結果ではローカルリポジトリのmasterブランチがリモートリポジトリのmasterブランチに同期されたことがわかります。

また、Bitbucket上のリポジトリにアクセスすると、masterブランチの状態が変わっていることも確認できます（**図12**）。

ちなみに、1人での作業においては、基本的にpushでエラーが発生することはありませんが、1人作業でも複数環境で同じリモートリポジトリにpushしていた場合は後述するようなケース（36ページ）でエラーが発生する可能性があります。



ブランチを切って1人で運用

1人の開発作業においても、1つの作業環境において複数の作業を並行して行うこともあります。たとえば、メインの開発を進める中で小さなバグ修正を行うといった場合です。

このような状況に対しては、Gitのブランチ機能が有効です。

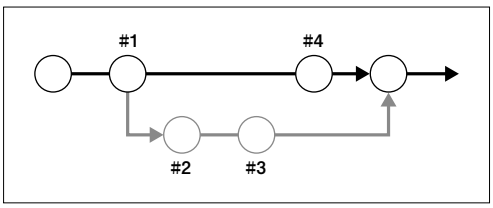
ブランチを利用すれば、現在作業中のファイル群をいったん退避させ、差し込まれた優先すべき作業を先行して行うことができます。優先作業が完了したら、退避させていたファイル群をもとの状態に戻し、そこに優先作業したファイルを統合することもできます（**図13**）。



ブランチを作って作業環境の状態を切り替える

Gitでブランチを作成するには、git branchコマンドを利用します。

▼図13　ブランチとは枝を意味し、作業フローを複数の開発ラインに分岐できる。その際、作業対象となるブランチをチェックアウトする




```
$ git branch develop
```

このようにコマンドに続けてブランチ名を記述して実行します。ここでは、別ライン開発用として「develop」というブランチ名を指定しています。これにより、現在の作業状況がdevelopブランチとして複製されます。作業中の（コミットしていない）状況でブランチを作成すると、最後にコミットした状況でブランチが作られる点に注意してください。

ブランチの確認と切り替え

ブランチを作成したら、まずはブランチの状況を確認してみましょう。オプションなしでgit branchコマンドを利用します。

```
$ git branch
develop
* master
```

先ほど作成したdevelopと、デフォルトで作成されるmasterという2つのブランチが存在することが確認できます。また、masterブランチの行頭に「*」が付いており、こちらがチェックアウト（現在の作業対象）となっていることがわかります。

作業対象をdevelopブランチに切り替えましょう。git checkoutコマンドを利用します。

```
$ git checkout develop
Switched to branch 'develop'
```

コマンドに続けてブランチ名を指定することで、作業対象ブランチを切り替えることができます。



別のブランチの作業状況をマージしよう

ここで、git checkoutコマンドでmaster、developを切り替えながら作業を並行して行っているという状況を想定しましょう。ブランチ

の切り替えによって、各ブランチで管理しているファイル群に修正が加えられています。これらの並行作業を確認しつつ、最終的には1つの作業結果として統合する必要があります。

2つのブランチの状況を確認

まずは現時点で2つのブランチがどのようになっているのかを確認します。現時点ではmasterブランチがチェックアウトされている状態とします。git logコマンドを利用します。

```
$ git log --all --graph --pretty=oneline
```

「--all」オプションですべてのブランチを指定、「--graph --pretty=oneline」オプションでブランチの分岐状況をわかりやすくするようにしています（**図14**）。

この図では、一番下から時系列にコミット状況が表示されています。まず最初にinitialize repositryという初回コミットがあり、その後「#1」が付いたコミット、その後ブランチが作成され、「#2」「#3」が付いたコミットを作成。その後、再度ブランチが切り替えられて「#4」が付いたコミットが行われたことがわかります。

2つのブランチを統合する

現時点ではdevelopブランチでの作業は完了し、masterブランチがチェックアウトされているとします。これら2つのブランチを統合しましょう。

git mergeコマンドを利用します。このコマンドは、現在チェックアウトされているブランチにほかのブランチをマージするコマンドです。コマンドに続けて、マージしたいブランチ名を記述します。

このコマンドの実行結果は4種類あります。それぞれを見てみましょう。

Already-up-to-date

developブランチで何も作業していないような状況では、masterブランチで何か作業を進めていたとしてもマージする際に何も変える必要がないため、

```
$ git merge develop
Already up-to-date.
```

となります。「すでに最新状態です」という意味になります。

FastForward

developブランチを作成して修正作業を行ったあと、masterブランチでは何も作業していない、つまりdevelopブランチ作成以前の状態のまま何も変わっていない状況では、

```
$ git merge develop
Updating d08954a..c364e08
Fast-forward
index.php | 2 +-
show.php | 6 +++---
2 files changed, 4 insertions(+), 4 ➥
deletions(-)
```

となります。developブランチのコミット状況がそのままmasterブランチに反映されるだけとなります。結果的に、チェックアウトされているmasterブランチにはdevelopブランチで追加されたコミットが反映され、最新のコミット

▼図14　ブランチの分岐状況を確認する

```
$ git log --all --graph --pretty=oneline
* e2e8b4e57e5bd968e8e11bae80fc7b1ff54efdb3 fix #4
| * c364e08095cff972f85169e6e8996e9627e07993 fix #3
| * 132fe7fb414a6b7c4981e25e1af40e5913c08c7a fix #2
|/
* d08954a7495fc078fe2f828eefd3652adcbfa02d Bug Fix #1
* d2172ab401d845c447aac298db51c0a2cc64e7d5 initialize repositry.
```

状態になります。

AutoMerge（オートマージ）

master/developの2つのブランチで異なる作業を行ったが、幸いにも同じファイルの同じ個所を修正することなく、単にお互いの違いを統合すればよい、という状況では、**図15**のようになります。マージする際に、とくに問題となる個所がないため、Gitが自動的に2つのブランチを統合し、さらにgit add、git commitし、マージ後の新しいコミットまで自動的に作成します。

Conflict（コンフリクト）

今、統合しようとしているmaster/developの2つのブランチで、同じファイルの同じ個所に対して異なる修正を行っていた場合、**図16**のようになり、マージ作業は中断され、ブランチの統合は行われません。

このようなコンフリクトが発生すること自体が悩ましい問題ですが、Gitがなければこの問題に気づかずに上書き、先祖返りしてしまうかもしれません。Gitを利用することで、問題個所を明示的に確認でき、さらには解決方法を検討して対応することが可能となります。

次項で、このコンフリクト対応についてみていきましょう。



コンフリクトの発生と解決方法

Gitのマージ作業で問題となるのがコンフリクトです。コンフリクトとはマージしようとする2つのブランチ間で「同じファイルの同じ個所」に対して「異なる修正を行っている」ために発生します。この場合、どちらの修正内容を「正」とするかが問題となりますが、この問題を明示的に示してくれることこそが、Gitを利用する最大のメリットの1つとなります。Gitがなければこれらの違いに気づかないかもしれません。その結果、大きな不具合を生じさせる可能性もあります。

Gitを利用していれば、2つの作業ラインで異なる修正を行っていることを明示的に示してくれます。当たり前のことですが、Gitは明示的に示すだけであり、どちらが正しいのかは作業者自身で判断しなければなりません。

コンフリクト個所を特定し、適正に修正する

まずは、どのファイルの、どの個所がコンフ

▼図15　ブランチが問題なく統合できた場合

```
$ git merge develop
Merge made by the 'recursive' strategy.
index.php | 2 +-
show.php | 6 +++---
2 files changed, 4 insertions(+), 4 deletions(-)
$ git log --all --graph --pretty=oneline
* 9862f1b6b23bb9e1cf80730b90c369cbd2f90e04 Merge branch 'develop'
|\
| * c364e08095cff972f85169e6e8996e9627e07993 fix #3
| * 132fe7fb414a6b7c4981e25e1af40e5913c08c7a fix #2
* | e2e8b4e57e5bd968e8e11bae80fc7b1ff54efdb3 fix #4
|/
* d08954a7495fc078fe2f828eefd3652adcbfa02d Bug Fix #1
* d2172ab401d845c447aac298db51c0a2cc64e7d5 initialize repositry.
```

▼図16　コンフリクトによりマージが中断された場合

```
$ git merge develop
Auto-merging index.php
CONFLICT (content): Merge conflict in index.php
Automatic merge failed; fix conflicts and then commit the result.
```

リクトとなっているのかを確認しましょう。まずはgit statusコマンドを利用します（**図17**）。

この最下行の「both modified: index.php」から、index.phpが「Unmerged paths」として記録されていることがわかります。そこで、index.phpをテキストエディタで開いてみます。すると、**図18**のような記述個所が見つかります。

「<<<<<<< HEAD …… =======」という個所はHEADつまりcheckout中のmasterブランチでの作業であり、「======= …… >>>>>>> develop」という個所はdeveopブランチでの作業です。2つの作業内容を確認して、どちらを採用するか検討して、修正を行いましょう。

具体的にはHEADが正しいのであれば、

```
$builder = new SQLBuilder(array(
"quote_name" => function($v){ return $v; },
"quote_value" => function($v){ return mysql_real_escape_string($v); },
));
include("show.php");
```

といったように、「<<<<<<< HEAD …… ======」や「======= …… >>>>>>> develop」な

▼図17　コンフリクトの個所を確認する

```
$ git status
On branch master
Your branch is ahead of 'origin/master' by 2 commits.
(use "git push" to publish your local commits)
You have unmerged paths.
(fix conflicts and run "git commit")
Changes to be committed:
modified: show.php
Unmerged paths:
(use "git add <file>..." to mark resolution)
both modified: index.php
```

▼図18　index.phpのコンフリクト個所

```
$builder = new SQLBuilder(array(
"quote_name" => function($v){ return $v; },
<<<<<<< HEAD
"quote_value" => function($v){ return mysql_real_escape_string($v); },
=======
"quote_value" => function($v){ return addslashes($v); },
>>>>>>> develop
));
include("show.php");
```



コンフリクトが解決できない場合

マージ処理でコンフリクトが発生したとき、すぐに解決できないこともあります。そのようなとき、「<<< …… === …… >>>」のようなコードが入ったままになっていて困ることもあります。

そのような場合には、

```
$ git merge --abort
```

というコマンドを実行することで、マージ作業を始める以前のリポジトリの状態に戻ることができます。

元の状態に戻ったうえで、再度作業を行い、最終的に最適な状態になってからマージ処理（場合によってはコンフリクトの解消）を行います。

どの不要な記述を削除して、正しいコードのみの状態に編集するということです。

コンフリクト個所が複数であれば「git add index.php」など、1ファイルずつ確認・修正を行ってコンフリクト状態を解消していきます。そして、すべての個所が解消できたのち、

```
$ git commit -m"Merge branch 'develop'"
```

というコマンドを実行することで、マージ作業を完了します。

最終的に、git logコマンドでマージされて作成されたコミットを確認しておきましょう（**図19**）。



Bitbucket経由で複数人で共同作業する

Bitbucket上のリモートリポジトリを「正」として、複数人で共同作業する場合を想定してみましょう。この場合、複数の作業者がリモートリポジトリに対してpushするため、コンフリクトが発生する可能性が常につきまとうことになります。

pushでエラーになるケース

リモートリポジトリに対するpushを行った際に、**図20**のようなエラーが発生することがあります。

この中に「! [rejected] master -> master

▼図19　マージ後のコミットの表示

```
$ git log --all --graph --pretty=oneline
* 5bb9a69f5cc8b68cfe7be246b223051e47053d47 Merge branch 'develop'
|\
| * 294567866f9c11ac485aac1ea7071fe255670b65 fix #5 by addslashes
| * c364e08095cff972f85169e6e8996e9627e07993 fix #3
| * 132fe7fb414a6b7c4981e25e1af40e5913c08c7a fix #2
* | 4a57e893eca0b1bc8b0c5bb1af212d76ae61e825 fix #5 by mysql_real_escape_string
* | e2e8b4e57e5bd968e8e11bae80fc7b1ff54efdb3 fix #4
|/
* d08954a7495fc078fe2f828eefd3652adcbfa02d Bug Fix #1
* d2172ab401d845c447aac298db51c0a2cc64e7d5 initialize repositry.
```



過去の状態を取り戻す

1人作業において、Gitをバックアップとして利用している場合、任意の過去の状態に戻すテクニックも重要です。現在の作業とは別のアプローチを試したいといった場合、過去のコミットをベースとして別のブランチを作成して作業するといった並行作業もできます。

任意のコミットをベースとして新しいブランチを作成するには、

```
$ git branch master_test_1 d08954a7495fc078fe
```

といったコマンドを実行します。この場合、「master_test_1」が新しいブランチ名、それに続けてベースとしたコミットIDを入力します。

新たなブランチを作成したら、それをチェックアウトして作業します。作業が完了し、ブランチ自体が不要となった場合は、

```
$ git branch -D master_test_1
```

というコマンドを実行することで、ブランチを削除できます。

▼図20　push時のエラー

```
$ git push origin master
Password for 'https://sample-git-user@bitbucket.org':
To https://sample-git-user@bitbucket.org/sample-git-user/sample.git
! [rejected] master -> master (non-fast-forward)
error: failed to push some refs to 'https://sample-git-user@bitbucket.org/sample-git-user/
sample.hint: Updates were rejected because the tip of your current branch is behind
hint: its remote counterpart. Integrate the remote changes (e.g.
hint: 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
```

▼図21　ローカルリポジトリの状況を確認する

```
$ git log --pretty=oneline
1f3328db3922071b9bb1e6811d8573c9db5330b7 fix #6

d08954a7495fc078fe2f828eefd3652adcbfa02d Bug Fix #1
d2172ab401d845c447aac298db51c0a2cc64e7d5 initialize repositry.
```

(non-fast-forward)」とあるため、Bitbucket上のリモートリポジトリのmasterブランチがローカルのmasterブランチの過去の状態ではなく、ほかの作業者によって異なるコミットが追加された状態であることが予想されます。

そこで、まずはローカルリポジトリの状況を`git log`コマンドで確認します（**図21**）。

さらにBitbucketにアクセスしてリモートリポジトリの状況も確認します（**図22**）。

pushの前にpullを行ってFastForwardな状況にする

先ほどの実行結果を比較すると、ローカルリポジトリ側には「#4」「#5」が付いたコミットが存在しないことがわかります。そこで、リモートリポジトリのmasterブランチをローカルリポジトリに同期することで、ローカルリポジトリをFastForwardな状況にします。リモートからローカルへの同期処理には、`git pull`コマンドを利用します（**図23**）。

▼図22　Bitbucket上のリモートリポジトリの状況。「#4」「#5」が付いたコミットが追加されていることがわかる



▼図23　リモートからローカルに同期+マージ処理を行う

```
$ git pull origin master
Password for 'https://sample-git-user@bitbucket.org':
From https://bitbucket.org/sample-git-user/sample
* branch master -> FETCH_HEAD
Merge made by the 'recursive' strategy.
builder.php | 2 +-
index.php | 2 +-
2 files changed, 2 insertions(+), 2 deletions(-)
```

これにより、ローカルリポジトリにリモートリポジトリのmasterブランチの最新状況が同期されたうえ、ローカルリポジトリの「#6」が付いたコミットが最新となり、リモートリポジトリ上のmasterの最新コミットは過去の状態になります。この状態であれば、ローカルからリモートへのpushは「FastForward」となり、エラーが発生することなくマージが完了します。

とくに複数人で共同作業している場合には、この現象は当然のように発生します。重要なことは、作業開始時にpullして、リモートとローカルのリポジトリに差分がないか確認し、差分があれば同期しておくこと。そして、作業完了時に必ずpushして、リモートリポジトリを常に最新の状態に保っておくことを忘れないようにすることです。SD



ある1つのファイルだけ過去の状態を確認したい

開発作業を進めている際、任意のファイルの過去の状態を確認したいといったときには、

```
$ git show d08954a7495fc078fe:index.php
```

といったコマンドを実行します。「git show」に続けて「コミットID:ファイル名」を入力して実行することで、該当ファイルの内容を確認できます。

また、

```
$ git diff d08954a7495fc078fe -- index.php
```

といったように「git diff コミットID -- ファイル名」と指定することで、コミットIDの時点と現時点での差分を確認することもできます。



全行コンフリクトを避けるための文字コードと改行コードの設定

改行コードや文字コードなどを変更すると、ファイルの全行が変更されたとみなされてしまい、ほかのブランチなどで同じファイルを編集していると、ファイル全体がコンフリクト個所と認識されてしまいます。こういった状況は、マージ作業に非常に苦労することとなるため、絶対に避けるべきでしょう。

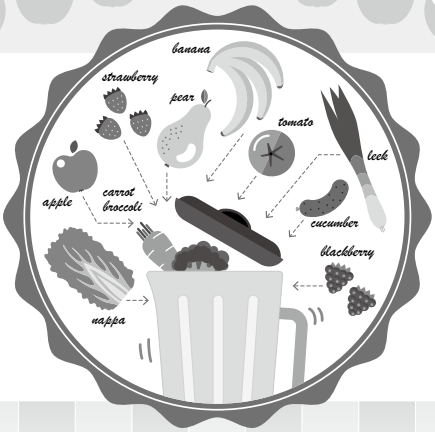
編集用のエディタで改行コードや文字コードの自動変換設定を行っているような場合、現在作業しているGit管理下のファイルに適用すべきコードと整合が取れているかどうか、事前に確認しましょう。また、git addでコミット用のステージにセットする前に、git diffで意図していない個所が差分として認識されていないかどうか確認し、問題があればコミットする前にしっかりと解決するようにしましょう。



第3章

構造を知るとGitの真価がよくわかる

Gitの内部構造とコマンドの作用の背景



本章では、リポジトリやコミットのデータ構造について解説します。Gitコマンドがリポジトリ内のデータ構造にどのような作用を及ぼすかを知ると、Gitの真価が理解できます。また、分散バージョン管理の転送メカニズムについても解説します。

AUTHOR 豊沢 泰尚（とよさわ やすたか）
㈲シェアリングシード
AUTHOR 久保 靖資（くぼ やすし）
㈱エクスパ



Gitは分散バージョン管理システム

Gitは、2005年、Linuxカーネルを開発していたコミュニティ発祥の優れたバージョン管理システムです。その特徴は、

・効率的な履歴データの管理により、高速な動作を実現
・ブランチ機能により、効率的な同時並行作業、協働作業を実現

している点にあります。

ここでは、Gitバージョン管理の基盤となるリポジトリやコミットの内部的なデータ構造について触れていきます。内部構造を知ること、さらにGitコマンドがリポジトリ内のデータ構造にどのような作用を及ぼすかを理解することによって、Gitの真価が初めて理解できます。

また、分散バージョン管理の「分散」がどのような転送メカニズムによって支えられているかについても触れます。リポジトリ間の関係性を正しく理解することで、Gitの恩恵を余すところなく受けることができるはずです。

なお、本章は初心者向けにGitの理解を深めるためにGitの内部構造に触れており、前章までに触れたGitの操作の背景を説明する内容となっています。



Gitリポジトリの内部的構造を知る

Gitのコア部分は、シンプルなKVS（キーバリュー型データストア）です。

Gitで作成するリポジトリは、コアとなるKVSと、KVS要素への参照や設定ファイルによって構成されています。また、コミットも、KVSの中に格納されています。

Gitはファイルシステム上のディレクトリ構造をファイルストレージとしてそのまま記録するのではなく、ツリー・ブロブオブジェクト（オブジェクトストレージ）として記録しています。これにより、分散するリポジトリ間での転送も効率的に行えます。



作業リポジトリの内部構造

前章で解説した作業リポジトリを例にして、Gitリポジトリがどのような構造になっているかを見てみましょう。クローンによって作成されたGit管理下のディレクトリには.git/というディレクトリが存在しています。この.git/配下のファイル群がリポジトリの実体となります。

なお、ここでは第2章で説明したGitの基本的な運用に直接関係するファイルを抜粋して記載しています。さらに高度な使い方に関係するファイルが本来多数含まれていますが、省略し



Gitのコマンドは大きく２種類ある

Gitコマンドは、大きく「磁器系（Porcelain）コマンド群」と「配管系（Plumbing）コマンド群」の2種類に分けることができます。

磁器系（Porcelain）コマンド群には、第2章で紹介したような基本コマンドのほか、checkout、branch、remoteなど、一般的に使用されるユーザフレンドリーなGitコマンド群です。

これに対して配管系（Plumbing）コマンド群は「バージョン管理システムを構築するため」のコマンド群であり、cat-file、hash-object、update-indexなど、GitのコアであるKVSを操作するものです。分散バージョン管理システムを支えている内部構造に直結するGitコマンドであり、内部構造を把握してなければ理解することが難しいものです。Gitのしくみをほかのシステムと密に連携させるうえでは便利なコマンドが含まれていますが、一般的に利用する機会はあまりないと言えるでしょう。

ただ、一般的に利用するコマンド群よりも低レベルのGitコマンドがあることを知ることで、Gitの真価をより深く知ることができるはずです。

てあります。

図1が.git/配下のファイル構成です。このうち、Gitの基本的な運用に直接関係するファイルに関して**表1**にまとめています。

参照の内部データ構造

Gitで作成・管理するブランチやリモートリポジトリとの同期をとるためのリモート追跡ブランチ、タグといった情報は、いずれも参照と呼ばれる情報です。

表1に示したとおり、/refs以下に配置されたコミットIDが記述された単純なテキストファイルが実体であり、各ブランチやタグがどのコミットIDを参照しているかを表現しています。

ブランチ、リモート追跡ブランチ、タグの3種類の参照の特徴も**表1**にあるとおりで、ブランチはチェックアウト（作業対象とする）できる参照。リモート追跡ブランチは、リモートリポジトリ上のブランチが最後に同期した時点での参照先を保持しています。

タグは特別な機能を持たず、単純にコミットに別名を持たせるといった用途で利用できるものです。

▼図1　Git管理下のディレクトリ（git/配下のファイル構成）

```
.git
├── config
├── HEAD
├── index
├── logs
├── objects
│   └── 63
│       └── 7be215174b38316416a2c3056d639b4a2c5e2a
└── refs
    ├── heads
    │   ├── master
    │   └── develop
    ├── remotes
    │   └── origin
    │       └── master
    └── tags
```

コミットの内部構造

Gitでファイル変更の履歴を記録しているのがコミットです。では、コミットにはどのような情報が記録されているのでしょうか。それを知るために、`git log`コマンドの実行結果をたどってみましょう（**図2**）。

git logコマンドの実行結果には、コミットIDのほか、コミットしたユーザ（Author）、記録日時、コミットメッセージが表示されています。通常、Gitコマンドでユーザが作業するコミットとは、このようなデータ構造となってい

るわけです。

しかし、実際にはそれ以前のコミットからの変更点（差分情報）などの履歴も保存されているはずです。対応するコミットオブジェクトの内部を覗(のぞ)いてみましょう。git cat-fileコマンドを利用します（**図3**）。

git cat-fileコマンドに、内容を読みやすいように整形して表示する「-p」オプションをつけたあと、コミットIDを指定して実行します。

実行結果には、git logコマンドの実行結果にも表示されていた情報以外に、treeやparentという情報があることがわかります。

treeは、コミットに対応するツリーオブジェクトのハッシュです。ツリーオブジェクトにはコミットした時点でのファイルの状態が記録されています。

parentは、1つ前のコミットIDです。git logコマンドはこのparent情報を利用して過去にさかのぼって履歴を列挙しているのです。つまり、1つ前のコミットへのリンク構造であるため、

▼表1　gitリポジトリの主な構成ファイル

ファイル名	説明
/HEAD	HEADの実体。現在チェックアウトされているブランチ名が記述されたテキスト。コミットIDではなくブランチ名が記載されている
/refs/heads/ブランチ名	ブランチの実体。ブランチの参照しているコミットIDが記述されたテキスト。ただコミットIDが記載されているのみで、headsという名前からもわかるように、HEADが参照できるのはここにある情報のみとなる。つまり、チェックアウトできるのは通常のブランチのみであることを意味している
/refs/remotes/リモート名/ブランチ名	リモート追跡ブランチの実体。リモート追跡ブランチの参照しているコミットIDが記述されたテキスト。リモートリポジトリのURLなどは記載がなく、通常のブランチと同じで、リモート追跡ブランチも通常のブランチと同様に扱えることを意味している
/refs/tags/タグ名	タグの実体。タグの参照しているコミットIDが記述されたテキスト
/index	インデックスの実体。現在ステージされている、コミット前の情報が記述されたバイナリファイル
/config	Gitリポジトリの設定が記載されているテキストファイル。どのようなリモートリポジトリがあるか、URLもここに記載されている
/logs	各ブランチ別でのGit操作の履歴が記載されている。コミット履歴が記載されているわけではない。名前から混同しそうだがgit logコマンドの結果には関係ない
/objects/	Gitのコアの部分であるKVSのデータ構造が配置されている。コミットの実体もここにあり、KVSのレコードであるオブジェクトと呼ばれる情報の一種である

▼図2　git logコマンドの実行結果

```
$ git log
commit 637be215174b38316416a2c3056d639b4a2c5e2a
Author: DevelopmentTeam <dev@localhost>
Date: Sun Aug 6 23:00:54 2017 +0900
fix #6 move lib files
```

▼図3　git cat-fileコマンドの実行結果

```
$ git cat-file -p 637be215174b38316416a2c3056d639b4a2c5e2a
tree 3f5cd29c31b2f635bd5d5cf4462e64667fa3f3c6
parent 4a57e893eca0b1bc8b0c5bb1af212d76ae61e825
author DevelopmentTeam <dev@localhost> 1502028054 +0900
committer DevelopmentTeam <dev@localhost> 1502028110 +0900
```

次のコミットの特定は困難であるということです。実際、どのブランチからもたどれなくなったコミットを探し出すことは難しく、このような状況になったコミットを「迷子になっている」と表現することもあります。

ツリーオブジェクトとは

先ほど述べたツリーオブジェクトは、各コミットに対応するファイルの状態を記録したものですが、その内容はどうなっているのでしょうか。ツリー情報自体もオブジェクトであるため、git cat-fileコマンドでその中身を見ることができます（**図4**）。

git cat-fileコマンドに内容を見やすく整形して表示する「-p」オプションをつけたあと、コミットIDを指定して実行します。

この実行結果を見ると、複数のオブジェクトのIDが並んでいることがわかります。各オブジェクトはblob、treeといったタイプが示されており、tree（ツリーオブジェクト）がディレクトリに、blob（ブロブオブジェクト）がファイルに対応しています。

これはOSのファイルシステム同様のディレクトリ構造を表現しているのですが、ファイルの名前や属性がディレクトリにあたるツリーのほうに記載され、ブロブオブジェクトにはファイルのコンテンツデータだけが格納されているという点がOSのファイルシステムとは異なります。

ブロブオブジェクトとは

続けてブロブオブジェクトについても、その中身を見てみましょう。やはりgit cat-fileコマンドを利用します（**図5**）。

この実行結果を見ると、単にファイルの内容が表示されています。つまり、コミットは以前の状態との差分の情報を記録しているわけでは



オブジェクトのID（ハッシュ）とは何なのか

Gitのコアである KVS に対してデータが登録される際、登録する情報のハッシュをとったものがIDとなります。このID（ハッシュ）をKeyとして利用することで、KVSに登録されているデータを取得できるわけです。

ハッシュは、同一のコンテンツであれば、同一のIDになるという性質を持っているため、同じデータが別IDで重複して登録されることはなく、効率的にデータが管理されていることがわかります。

▼図4　ツリーオブジェクトに対してのgit cat-fileコマンドの実行結果

```
$ git cat-file -p 3f5cd29c31b2f635bd5d5cf4462e64667fa3f3c6
100644 blob 4e8798ae51059e3b488c0a1392facb7d8ca82a9a index.php
040000 tree 0166e2345bf223909594d7d4635d82a8a07e9ecb lib
100644 blob eb7bc5e78c23c81589f3b0e160392f1ecc0dc33c show.php
```

▼図5　ブロブオブジェクトに対してのgit cat-fileコマンド実行結果

```
$ git cat-file -p 4e8798ae51059e3b488c0a1392facb7d8ca82a9a
<?php
require_once("lib/builder.php");
$query_json = $_POST['query_json'];
$query = json_decode($query_json, true);
$builder = new SQLBuilder(array(
"quote_name" => function($v){ return $v; },
"quote_value" => function($v){ return mysql_real_escape_string($v); },
));
include("show.php");
```

なく、コミットした時点でのファイルの内容そのものを記録しているのです。

コミットの内部構造とは

コミットについて追ってきましたが、結局のところコミットとは何なのでしょうか。

コミットはGitのコアKVSの中で、コミット、ツリー、ブロブという3種類のオブジェクトによって表現されています（**図6**）。そして、ツリーがファイルシステムの構造、ブロブがファイルの内容を表現しているのです。

つまり、Gitは以前の状態との差分の情報ではなく、コミット時点のディレクトリ構造やファイルの内容そのものを記録しているのです。同時に、GitのコアKVSのしくみにより、変更されていない情報が重複して登録されることはありません。これがGitコミットによるバージョン管理の本質です。

ただし、このしくみであるがゆえに、容量の大きなファイル（たとえばWebサイトデータを管理する場合の画像データなども含めて）をリポジトリに登録した場合、ファイルに少しでも変更を加えてコミット登録を行うたびに.git/内のファイルサイズが肥大化していくことに注意が必要です。

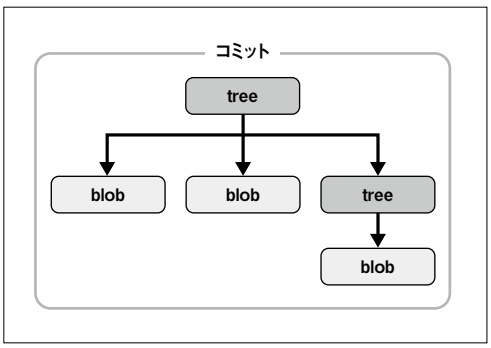


分散バージョン管理のしくみと転送メカニズム

Gitによる分散バージョン管理のしくみと転送メカニズムとは、どのようなものなのでしょうか。

ここまでで見てきたリポジトリ、コミットの内部構造を踏まえると、理解は容易であるはずです。リモート／ローカルのリポジトリ間の通信によって行われているのは「オブジェクトの同期と、参照の移動」だけなのです。

▼図6　ツリーがファイルシステムの構造、ブロブがファイルの内容を表現している





リモートリポジトリの情報はどこにあるのか

Gitのpushコマンドでリモートリポジトリに同期する際、リモートリポジトリを「origin」と指定します。originはデフォルトで定義されているリモートリポジトリの別名ですが、実際にはどのような内容なのでしょうか。`git remote show`コマンドでorigin、つまりリモートリポジトリへのアクセス情報を確認できます（**図7**）。

実行結果には、Fetch URLとPush URLという2つのURL情報が表示されます。これは、



リモートリポジトリとは何か

狭義には、ネットワーク上のサーバにあるベアリポジトリを意味します。ベアリポジトリとは「HEADと作業ツリーがなく、コミットの作成ができない」リポジトリです。

Gitには「HEADに参照されているブランチに対してのpushは受け入れられない」というルールがありますが、ベアリポジトリにはHEADがないため、すべてのブランチに対してのpushを受け入れることができるという特性があります。このためpush先となるリポジトリに適しているわけです。

また、広義にはリモートリポジトリは作業しているリポジトリから見たそのほかのリポジトリのことを意味します。Bitbucketのようなサービス上にあることもあれば、同一マシン上であってもよく、作業リポジトリ自体もpushの対象にできます。

本特集では広義の意味でこの用語を用いています。

リモートリポジトリと通信するためのURL情報です。

ちなみに、この情報は.git/configに記述されています。catコマンドで.git/configの内容を確認してみましょう（**図8**）。fetch = 以降に記述されているのは参照仕様（Refspec）という情報で、git pullの動作を決める設定になります。



リモートリポジトリへのコミット履歴情報の送信を行うgit pushのしくみ

次に、リモートリポジトリへコミット履歴情報を送信するpushのしくみを見てみましょう。git pushコマンドの実行結果を確認します（**図9**）。

実行結果を見ると、まずはオブジェクトの転送が行われていることがわかります。これは、作業リポジトリには存在しますが、リモートリポジトリには存在しないオブジェクトの転送です。オブジェクト自体は同じID（ハッシュ）であれば必ず同じ内容ですので、同期は安全に行うことができるわけです。

続けて、リモートリポジトリ上のブランチの参照先を、作業リポジトリ上の同名のブランチの参照先に合わせています。参照先を合わせるだけであり、マージを行うことはできない点に注意が必要です。作業リポジトリ上の.git/refs/heads/masterの内容に合わせて、リモートリポジトリ上の.git/refs/heads/masterを書き換えようとします。

このとき、仮に対象ブランチがリモートリポジトリ上でチェックアウトされていたり、Fast Forwardにならない差分がある場合はマージが必要になる可能性があります。仮にこのような場合でもpushが受け入れられてしまったらどうなってしまうのでしょうか。マージは行えないので参照先だけが書き換わってしまうこと

▼図7　git remote showコマンドの実行結果

```
$ git remote show origin
* remote origin
Fetch URL: https://sample-git-user@bitbucket.org/sample-git-user/sample.git
Push URL: https://sample-git-user@bitbucket.org/sample-git-user/sample.git
HEAD branch: master
Remote branch:
master tracked
Local branch configured for 'git pull':
master merges with remote master
Local ref configured for 'git push':
master pushes to master (fast-forwardable)
```

▼図8　catコマンドで.git/configの内容を確認

```
$ cat .git/config
...
[remote "origin"]
    url = https://sample-git-user@bitbucket.org/sample-git-user/sample.git
    fetch = +refs/heads/*:refs/remotes/origin/*
```

▼図9　git pushコマンドの実行結果

```
$ git push origin master
Counting objects: 4, done.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (4/4), 401 bytes | 0 bytes/s, done.  ← オブジェクトを転送してリモートに書き込んでいることが読み取れます
Total 4 (delta 1), reused 0 (delta 0)
To https://sample-git-user@bitbucket.org/sample-git-user/sample.git
4a57e89..637be21 master -> master  ← リモートのブランチ参照先が変わっている
```

になり、リモートリポジトリ上のmasterが参照していたコミットが迷子になってしまう危険性があります。そこで、Git標準の設定により「危険なpushである」というエラーが発生し、pushは実行できないようになっているのです。



リモートリポジトリからのコミット履歴情報の受信と、マージを行うgit pullのしくみ

本章最後に、リモートリポジトリからコミット履歴情報を受信するpullのしくみを見てみましょう。

まず認識しておきたいのは、「pullとはfetchとmergeを組み合わせたコマンド」だということです。つまり、

```
$ git pull origin master
```

は、

```
$ git fetch origin
$ git merge origin/master
```

を実行することと同等です。

それぞれのコマンドを見てみましょう。

git fetch origin

git fetchコマンドは、git pushコマンドと対称的な作用を持っています。実行すると**図10**のように、まずはリモートリポジトリには存在しているが、ローカルリポジトリには存在しないオブジェクトを受信しています。

次にリモート追跡ブランチの参照先コミットIDを、同名のリモートリポジトリ上のブランチの参照先に合わせます。このとき、リモート追跡ブランチはorigin/masterという名前です。

▼図10　git fetchコマンドの実行結果

```
$ git fetch origin
remote: Counting objects: 4, done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 4 (delta 1), reused 0 (delta 0)
Unpacking objects: 100% (4/4), done.
From https://bitbucket.org/sample-git-user/sample
4a57e89..637be21 master -> origin/master
```

git pushの実行結果と同等の内容が表示されていることから、同等のことが行われていることがわかると思います。

git merge origin/master

```
$ git merge origin/master
```

これは先述したマージを行うコマンドです。

リモート追跡ブランチはorigin/masterという名前であるため、このようなコマンド指定となります。

これにより、現在ローカルリポジトリでチェックアウトされているmasterブランチに対して、origin/masterの参照するコミットをマージすることになり、結果的にリモートとローカルのリポジトリの同期が行われることになります。



まとめ

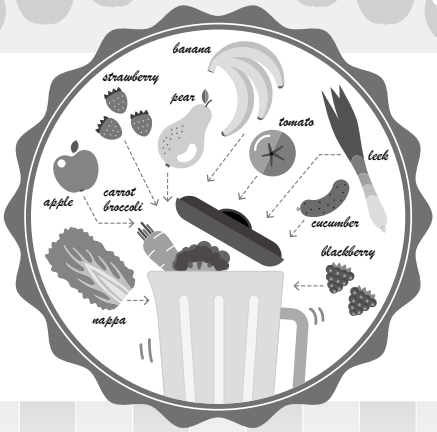
第2章、第3章を通してGitの基本的な使い方と、背後でGitがどのようなしくみで動作しているのかを説明しました。Gitはシンプルで強力なツールですので、工夫しだいで幅広いケースに応用することができます。今回は簡単な1人作業、連携作業について説明しましたが、Gitはさらに複雑で大規模なプロジェクトでも効果的に運用されている実績があります。まずはGitを使ってみて、少しずつ慣れ親しむ中で、より効果的な応用についても考えてみるとよいのではないかと思います。本記事がみなさんのGitを使い始めるきっかけや、より深く理解する助けになればうれしく思います。SD



第4章

強力プラグインgina.vim

実践！ エディタからのGit操作［Vim編］



Gitを利用するのはプログラミングのときです。プログラミングに常時使うエディタから、Gitがシームレスに使えたらとても便利です。本章では、エディタの代表として「Vim」をターゲットにし、Vim上から効率よくGitを利用するための手段を紹介します。

AUTHOR Alisue
㈱フィックスポイント
TWITTER @lambdalisue
BLOG http://lambdalisue.hatenablog.com/



VimからGitを使う意義

Gitはターミナルから実行することを前提として作られています。そのGitをVimから利用することに意義はあるのでしょうか？

Vimから利用する意義の1つに、Gitコマンドの利用頻度が挙げられます。たとえば、Gitは論理単位でコミットを分割することを推奨しています[注1]。そのため、Gitを利用するプロジェクトでは`git add`や`git commit`などを頻繁に実行することで、コミットの論理性を担保することが重要です。このように頻繁に実行するコマンドは、Vimからシームレスに呼び出せたほうが便利です。

また、Gitの機能をVimから呼び出すことにより、Vimの強力なテキスト操作機能が利用できます。たとえば、Gitには過去バージョンを表示する機能がありますが、ターミナルにて過去バージョンを表示するとVimの機能を使うことができません。その点、VimからGitを使えばオペレーションによるテキストオブジェクト単位での操作などが利用できます。

Gitには特殊な機能も多いため、すべてをVimから使うべきではありません。しかしながら、上述したとおりVimから使ったほうがよい機能が多いことも事実です。本章では、具体例とともにVimからGitを使う方法とその利便性を紹介します。なお、執筆環境は次のとおりです。

- macOS Sierra 10.12.6
- MacVim 8.0.839
- Git 2.11.0
- gina.vim v0.3.1
- vim-plug 449b4f1



簡易的にVimからGitを使ってみる

Vimは外部コマンドを次の方法で呼び出すことができます。

- **`:! {command}`** ……コマンドを実行し、結果を表示する
- **`:read !{command}`** ……コマンドを実行し、結果をカーソル位置に挿入する

この2つを用いると、機能的制限や使いにくさはありますが、簡単なGitの機能であれば直接使うことが可能です。サーバ上のVimなど、事前準備が難しい環境でも利用可能なため、まずは上記2つを用いてGitを利用する方法を紹

注1）Gitでの分散作業 - プロジェクトへの貢献 URL https://git-scm.com/book/ja/v1/Git-Gitでの分散作業-プロジェクトへの貢献

介します。

それでは、リポジトリの初期化から最初のコミットまでを実際に行ってみます。まずは適当な場所に空のディレクトリを作成し、そこでVimを起動してください[注2]。ここでは~/workspaceに作成したものとして話を進めます。

Vimを起動した後、最初に:!git initを実行してGitのリポジトリとして初期化してください。次に、fizzbuzz.vimとしてファイルを開き、**リスト1**の内容で保存してください[注3]。

ここで、現在のインデックス状態を確認してみます。git statusの結果にはファイル名などが含まれるため、バッファに出力します。**リスト2**に示された一連のコマンドを実行してください[注4]。

これにより、**図1**のように新規バッファにgit statusの結果が表示されます。

結果を確認したあとはfizzbuzz.vimにカーソルを合わせてgfでファイルを開いてください。その後、ステージングを行います。fizzbuzz.vimを開いていることを確認して、:!git add %としてステージングしてください[注5]。

ステージングを行った後、先ほどと同様にインデックスの状態を確認してください。fizzbuzz.vimが「Changes to be committed:」に表示されていればOKです。

最後に変更をコミットします。通常はgit commitを実行すると、エディタが起動してコミットメッセージの編集が可能です。しかし、Vimから呼び出した場合はエディタを開くことができません[注6]。しかたがないため、今回は:!git commit -m "最初のコミット"としてコミットメッセージを事前に与えてください。このあたりの使いにくさは、後述するプラグインを使うと解消されます。



本格的にVimからGitを使う

ここまでに見てきたように、プラグインを使わなくてもGitを使ったたいていの処理は可能です。しかしながら、より高度な使い方をする場合は、すでに存在するプラグインを導入するほうが簡単です。

Git用プラグインは数多く存在しますが、こ

注2） Vim内では「:pwd」で現在のディレクトリが確認できます。もし作成した空ディレクトリではなかった場合は「:cd {path}」で移動してください。

注3） このVim scriptを試しに実行してみたい場合は「:source %」とすることで可能です。

注4） 一連のコマンドを始める前に「qs」を押し、一連のコマンドが終わったあとに「q」を押すとマクロとして記録されます。マクロの実行は「@s」でできるため、一連のコマンドを再度実行する必要がなくお勧めです。詳細は「:help recording」を参照してください。

注5） Vimのコマンドに「%」を指定すると現在編集中のファイル名に展開されます。

注6） 正確にはGVimやMacVimなど、ターミナル外からVimを使っている場合は「$EDITOR」を開くことができないため、コミットメッセージを事前に与える必要があります。

▼リスト1　Vim scriptで書かれたFizz Buzz

```
for n in range(1, 50)
    if n % 3 == 0 && n % 5 == 0
        echo 'Fizz Buzz'
    elseif n % 3 == 0
        echo 'Fizz'
    elseif n % 5 == 0
        echo 'Buzz'
    else
        echo string(n)
    endif
endfor
```

▼図1　「git status」の結果をバッファに表示

```
[No Name]
On branch master

Initial commit

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        fizzbuzz.vim

nothing added to commit but untracked files present (use "git add" to track)
~
```

▼リスト2　git statusの結果をバッファに表示するコマンドセット

```
:enew
:read !git status
:0delete
:setlocal nomodified nomodifiable ↵
buftype=nofile
```

こでは筆者が作成を行っているgina.vim[注7]（以後gina）を紹介します。ginaはGitの機能全般を扱うプラグインです。なお、ginaはVim 8.0.0027もしくはNeovim 0.2.0以降に対応しています[注8]。

プラグインの導入

ここではvim-plug[注9]というプラグインマネージャーを利用します。プラグインマネージャーに頼らずに導入したい方は`:help add-plugin`を参考にしてください。

vim-plugは単一ファイルで構成されるプラグインです。導入は、https://raw.githubusercontent.com/junegunn/vim-plug/master/plug.vimをダウンロードし、plug.vimという名前で**表1**のディレクトリに保存します。その後.vimrcに**リスト3**の内容を追記してください。

追記後、**Vimを再起動**し`:PlugInstall`を実行すると、**図2**のようにginaがインストールされ、`:Gina`コマンドが使えるようになります。

インデックス状態の表示および変更

それでは、ginaの具体的な使い方を紹介していきます。まず、先ほど作成したfizzbuzz.vimのFizz Buzz本体を、**リスト4**のように関数に切り出してください。

次に`:Gina status`を実行してインデックス状態を表示してください。これにより`gina://workspace:status`というバッファに、**図3**のように現在のインデックス状態が表示されます[注10]。

次はステージングですが、ginaは**アクション**という機能によりバッファ内でさまざまな機能を使うことが可能です。まず、対象としたい行（fizzbuzz.vimが含まれる行）にカーソルを合わせてください。次に、対象行にて Tab キーを

注7） https://github.com/lambdalisue/gina.vim
注8） Vimのバージョン確認は「:version」で可能です。バージョンが古い場合は、新しいバージョンをインストールしてからお試しください。
注9） https://github.com/junegunn/vim-plug
注10） 「Untracked files:」に「.fizzbuzz.vim.swp」が表示されることがあります。これはVimが作成するスワップファイルなのでGitで管理するべきではありません。そのため「~/.gitignore」などに「*.swp」を追加し、無視するようにすると良いでしょう。

▼表1　vim-plugを保存するディレクトリ

OS	Vimの場合	Neovimの場合
Linux/macOS	~/.vim/autoload	~/.config/nvim/autoload
Windows	~¥vimfiles¥autoload	~¥AppData¥Local¥nvim¥autoload

▼リスト3　vim-plugを利用してgina.vimを有効化するための.vimrcの内容

```
" vim-plugでインストールするプラグインはcall ▽
plug#begin(...) から
" call plug#end() の間にPlug ... として記載する。
call plug#begin('~/.vim/plugged')

" gina.vimの指定
Plug 'lambdalisue/gina.vim'

" 上記したプラグインの初期化処理
call plug#end()

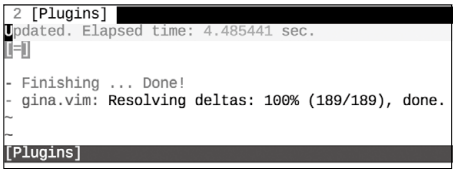
" vimdiffで横分割を使う設定
" Ginaはdiffoptの値で分割方向を決めるので、ここで指定する
set diffopt+=vertical
```

▼リスト4　Fizz Buzz本体を関数に切り出したfizzbuzz.vimの内容

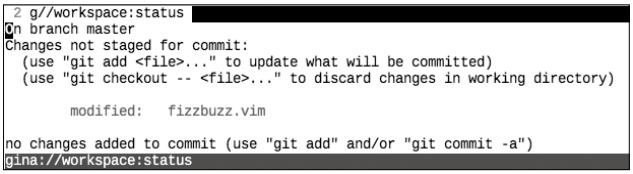
```
function! FizzBuzz(n)
    if n % 3 == 0 && n % 5 == 0
        echo 'Fizz Buzz'
    elseif n % 3 == 0
        echo 'Fizz'
    elseif n % 5 == 0
        echo 'Buzz'
    else
        echo string(n)
    endif
endfunction

for n in range(1, 50)
    call FizzBuzz(n)
endfor
```

▼図2　「:PlugInstall」の結果



▼図3　「:Gina status」の実行結果



押してください。するとaction:と下部に表示されるので、そこにstage[注11]と入力しEnterキーで確定してください。これでステージングアクションが実行されます。

このようにginaはさまざまな機能をアクションとして提供しています。ほかにどのような機能が提供されているか知りたい場合は、アクションが提供されているバッファにて?を押してください。これにより、実行可能なアクションの一覧が表示されます。

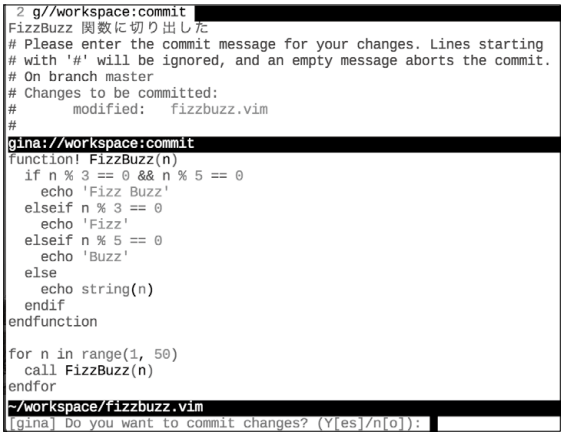


ステージングされた変更のコミット

先ほどはコミットメッセージをあらかじめコマンドに与えましたが、ginaを用いた場合はコミットメッセージの編集をバッファで行うことができます。fizzbuzz.vimがステージングされていることを確認した後、:Gina commitを実行してください。するとgina://workspace:commitというバッファが開くので、ここにコミットメッセージを記載し:wで保存してください。

記載したコミットメッセージに満足したあとは:qでバッファを閉じてください。すると**図4**のようにコミットするか否かを聞かれるため、そのままEnterを押します[注12]。なお:qではなく:wqで終了した場合は、確認なしでコミットが行われます。

▼図4　コミットバッファを「:q」で終了させた様子





変更点の部分適用

これで関数への切り出しが完了しました。しかし、このコードには現在次のような問題点があります。

- Vim scriptの関数において、引数参照には「a:」が必要だが付け忘れている（バグ修正）
- FizzBuzz関数はここでしか使わないのに、グローバル関数になっている（リファクタリング）
- Vim scriptの関数には「abort」を付けるべきなのに付いていない（お作法修正）

そこで、まずはすべての問題点を修正します。fizzbuzz.vimを**リスト5**のように修正してください。

それでは、この変更をステージングします。ただし、今回の変更には3種類の変更がすべて含まれており、論理単位ではありません。本来であれば1つずつ変更を適用していくべきでし

注11）「stage」は「index:stage」アクションのエイリアスです。また、アクション入力ではTabキーによる補完が効くため、アクション名を忘れた場合はTabキーを押して探すことが可能です。なお、バッファに定義されているアクション・エイリアスは「?」にて確認できます。

注12）「Y[es]/n[o]」のように片方が大文字で始まっている場合はデフォルト値を表しています。また[]で囲まれた文字列は省略可能なことを表しています。

たが、すでにワークツリーの状態はすべての変更が適用された状態です。このような場合、ginaのpatch機能が利用できます[注13]。patch機能はvimdiffを用いて視覚的に差分を表示しつつ、ステージングする変更を自由に編集するための機能です。そこで、今回はこのpatchを使って変更を3つに分割します。

まずは:Gina statusとしてインデックス状態を表示してください。次にfizzbuzz.vimにカーソルを合わせてppと入力します。これによりpatchアクションが実行され、**図5**のように3つのバッファが表示されます[注14]。

まずは最初の、バグ修正のステージングから行います。インデックスバッファの最初のハイライト個所（function! ...の行）にカーソルを合わせてdorを押してください。すると、インデックスバッファのハイライト個所がワークツリーバッファと等しくなります。このように、インデックスバッファに対しワークツリーの内容を簡単に適用することができます。なおHEADバッファから取得する方法など、より詳細な使い方は:help :Gina-patchを確認してください。

上述したマッピングや、インデックスバッファの直接編集などを駆使して、ステージングする内容を作成します。それでは、インデックスバッファの内容を**リスト6**のように「バグ修正だけを行った場合の内容」になるように修正してください。

この状態でインデックスバッファを:wで保存するとgina://workspace:statusが**図6**のように変化します。

ここでコミットを行うと、インデックスバッファの内容がコミットされます。このように

注13）ターミナル上では「git add -p」とすることで対話的に一部の変更のみをステージングすることができます。Ginaから「git add -p」を使いたい場合は「:Gina!! add -p」と「!!」を付加することで使えるようになります。

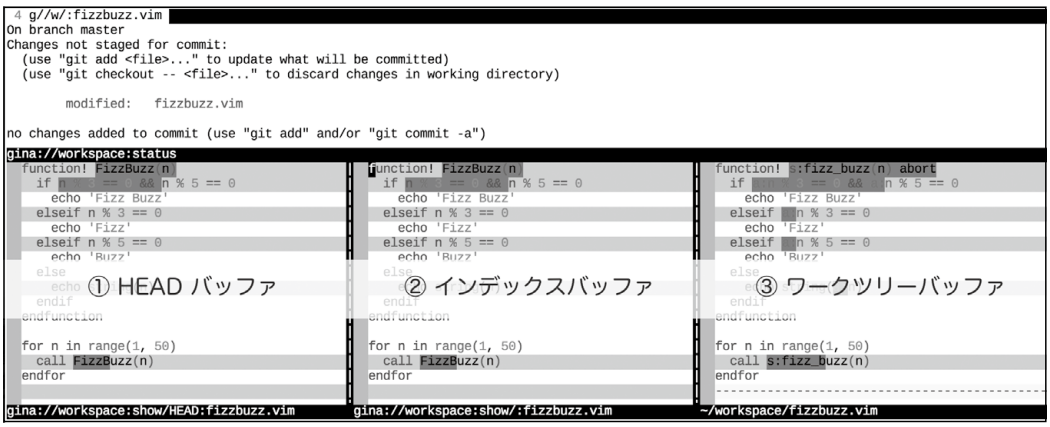
注14）利用頻度が高いアクションには、対応するマッピングが用意されています。また、ユーザが自由にカスタマイズすることも可能です。カスタマイズの詳細は「:help gina-custom」をご覧ください。

▼リスト5　すべての変更を適用したあとのfizzbuzz.vimの内容

```
function! s:fizz_buzz(n) abort
    if a:n % 3 == 0 && a:n % 5 == 0
        echo 'Fizz Buzz'
    elseif a:n % 3 == 0
        echo 'Fizz'
    elseif a:n % 5 == 0
        echo 'Buzz'
    else
        echo string(a:n)
    endif
endfunction

for n in range(1, 50)
    call s:fizz_buzz(n)
endfor
```

▼図5　「patch」アクション実行結果



▼リスト6　バグ修正だけを行った場合のfizzbuzz.vimの内容

```
function! FizzBuzz(n)
    if a:n % 3 == 0 && a:n % 5 == 0
        echo 'Fizz Buzz'
    elseif a:n % 3 == 0
        echo 'Fizz'
    elseif a:n % 5 == 0
        echo 'Buzz'
    else
        echo string(a:n)
    endif
endfunction

for n in range(1, 50)
    call FizzBuzz(n)
endfor
```

patchを利用するとワークツリーのコードを変更することなく、差分個所を視覚的に確認しながら部分的に適用することが可能です。

上記手順を参考にして、関数名の変更とabortの付加をそれぞれ別にコミットしてください。

コミット履歴の確認とアクションのドットリピート

ここまでに、何度かコミットを行ったので、リポジトリのコミット履歴を確認してみます。`:Gina log`を実行してください。すると`gina://workspace:log`というバッファにコミット履歴が表示されます。

次に、各コミットの詳細を表示してみます。まず、末尾行にて`show:preview`アクションを実行してください。すると、プレビューウィンドウに該当コミットの内容と変更点が表示されます[注15]。

それでは、2番目のコミット内容を表示してみます。再度`show:preview`アクションを実行しても良いのですが、代わりに該当行で`.`（ドット）を押してください。これにより、前回Tabキーを用いて実行したアクションがリピートされます。そのため、該当コミットの内容と変更点がプレビューウィンドウに表示されます（図7）。

このようにドットリピートを使うとアクション名の再入力が不要になります。カスタムマッピングを定義するほどではない場合に重宝します。

▼図6　変更を一部だけステージングした際の「:Gina status」の結果

```
 4+ g//w/:fizzbuzz.vim
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   fizzbuzz.vim

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   fizzbuzz.vim

gina://workspace:status
```

▼図7「:Gina log」の結果とドットリピートにより表示されたコミット詳細

```
 2 g//workspace:log
commit 72d21b4a858529ea188fcfb62caba1e5c9111e35
Author: lambdalisue <lambdalisue@hashnote.net>
Date:   Thu Aug 3 19:07:14 2017 +0900

    FizzBuzz 関数に切り出した

diff --git a/fizzbuzz.vim b/fizzbuzz.vim
index e5d20c7..2683a5a 100644
--- a/fizzbuzz.vim
+++ b/fizzbuzz.vim
@@ -1,4 +1,4 @@
-for n in range(1, 50)
+function! FizzBuzz(n)
   if n % 3 == 0 && n % 5 == 0
     echo 'Fizz Buzz'
gina://workspace:show/72d21b4 [Preview]
* c856818 - 空行の削除 19 hours ago <lambdalisue> (HEAD -> master)
* 3d055eb - abort の付加 19 hours ago <lambdalisue>
* 0a24d4b - 関数名の変更 (FizzBuzz -> s:fizz_buzz) 19 hours ago <lambdalisue>
* 43efd6c - バグ修正 (n -> a:n) 19 hours ago <lambdalisue>
* 72d21b4 - FizzBuzz 関数に切り出した 22 hours ago <lambdalisue>
* 961bec8 - 最初のコミット 24 hours ago <lambdalisue>
gina://workspace:log
```

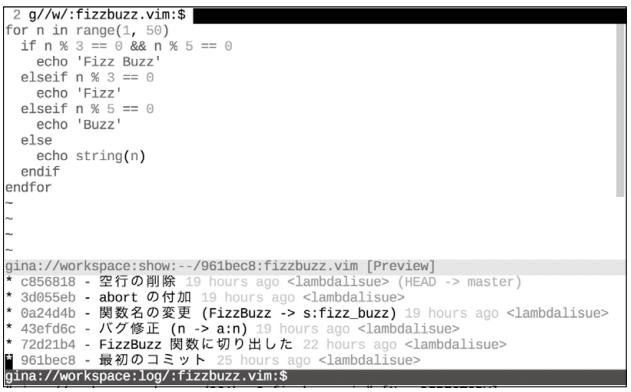
ファイルのコミット履歴と過去バージョン表示

コミット履歴の確認とともに多用されるのが、過去バージョンの表示です。Ginaでは特定ファイルのコミット履歴を表示する機能があり、過去バージョンの表示はここから簡単に行えます。

それでは、fizzbuzz.vimのコミット履歴を表示してみます。`:Gina log :fizzbuzz.vim`を実行してください（fizzbuzz.vimの前に`:`があることに注意してください）。すると`gina://workspace:log/:fizzbuzz.vim:$`という、先ほどのバッファ名にファイル名が付加されたバッ

注15）プレビューウィンドウのためフォーカスは移動しません。詳細は「:help preview-window」を参照してください。

▼図8 「:Gina log :fizzbuzz.vim」の結果と表示された過去バージョン



ファが開きます[注16]。今回はfizzbuzz.vimしかリポジトリに存在しないため、先ほどの内容と同じものが表示されています。ただし、このバッファには指定されたファイルのコミット履歴のみが表示されているため、たいていは:Gina logの結果とは異なるでしょう。

それでは、先ほどと同様に末尾行にカーソルを合わせてshow:previewアクションを実行してください。すると、今回はプレビューウィンドウに最初のコミット時のfizzbuzz.vimの内容が表示されます（**図8**）。

このように、ginaを使うことで任意のバージョン表示を簡単に行うことができます。また、割愛しますがdiffやcompareといったアクションにより、過去のバージョンとワークツリーの内容を比較することも可能です。

注16）「:$」という接尾字も付加されていますが、これは意図しないファイル名によるオートコマンド実行を避けるためです。



終わりに

VimからGitを利用する方法は千差万別です。ginaには今回紹介できなかったさまざまな機能があります。また、ginaではカバーしていない機能を提供するプラグインもたくさんあります。この章で紹介した内容にとどまらず、みなさん思い思いの「ぼくのかんがえた、さいきょうのギットかんきょう」を構築してみてはいかがでしょうか？SD



ショートフォーマットのすすめ

現在のインデックスを確認する際にはgit statusを実行しますが、この際に表示される内容は少々冗長です。ターミナルから実行した場合は余白が十分にあるため良いですが:Gina statusとした場合は、ウィンドウの高さが低いため一覧性がよくありません。そのため、--shortオプションを用いて、**図A**のようにショートフォーマットでインデックス状態を表示することをお勧めします。

各行の先頭にある**M**や**A**などが変更の種類を、文字の位置がステージング状態を表しています。このようにショートフォーマットではそれぞれのファイルの状態が各行に表示されるため、一覧性が高いです。なお、詳細は:help gina-misc-short-formatかman git-statusでご確認ください。

また、毎度オプションを指定してコマンドを実行するのは面倒なため、.vimrcに**リストA**のように書いておくと:Gina statusとするだけでオプションが適用されます。なお、詳細は:help gina-custom-commandを参照してください。

▼図A 「:Gina status --short」の結果

```
 M .travis.yml
M  README.md
A  autoload/gina/action/yank.vim
 M autoload/gina/command.vim
 M autoload/gina/command/blame.vim
 M autoload/gina/command/blame/formatter.vim
M  autoload/gina/command/branch.vim
```

▼リストA Gina statusコマンドのカスタム例

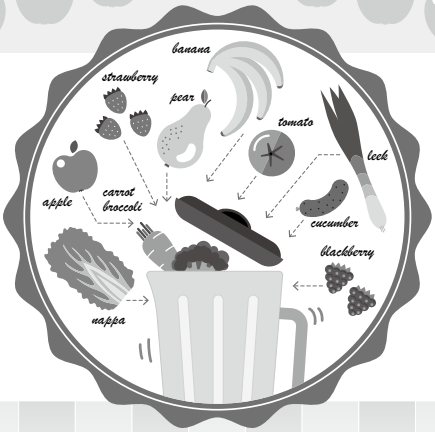
```
call gina#custom#command#option('status', ↵
'-s|--short')
```



第5章

GitからGitHubまで
エディタから操作して作業がはかどる！

実践！ エディタからのGit操作［Atom編］



「Atom」はGitHub謹製の人気あるテキストエディタです。バージョン1.18からは、GitとGitHubとの連携が強化され、よりGit操作がしやすくなりました。本章では、この連携機能を中心にAtomからGitやGitHubを使う方法と、それを補強するパッケージを紹介します。

AUTHOR 大竹 智也（おおたけ ともや）
BLOG http://d.hatena.ne.jp/tomoya/



AtomとGit

みなさんはどうやってGitを操作していますか？

Gitの操作には、コマンドラインインターフェース（CLI）以外にもGitHub Desktop[注1]などの専用クライアントソフトが存在しており、これらを利用する方も多いことでしょう。しかし、これ以外にもエディタを使って操作するという選択肢があるのはご存じでしょうか？

Atom、Emacs、Vimなど多くのエディタは標準でGitのサポートを行っています。また、Gitは指定したエディタでコミットメッセージが書けるしくみを持っており、git commitコマンドを実行したときに好きなエディタを起動することができます。

このようにエディタとGitは密接な関係を持っていますが、実際にどのような操作が行えるのか、また、エディタから操作するとどんなメリットがあるのかなど、まだ試したことのない方は疑問に思うことでしょう。

本章ではAtomを使ったGitの操作について解説していきます。具体的な操作法以外にも、エディタからGitを操作するメリットについても存分に解説していきますので、まだ、試したことのない方はぜひ一度試してみてください。

なお、本章で解説している動作確認は、mac OS 10.11.6のAtom 1.21.0-devになります。



AtomでGitを扱うメリット

筆者が考える、エディタからGitを操作する最大のメリットは次の2つに集約されます。

❶ターミナルや専用クライアントに切り替える必要がなく、作業に集中できる
❷実際のコード変更を確認しながらコミットが行えるため、ミスが発生しにくい

❶は、作業速度と効率を改善してくれます。もし、1つのファイルを編集してコミットしたいとき、エディタからターミナル、そしてコミットエディタとソフトウェアを切り替えるより、編集したエディタからそのままコミットを行ったほうが、圧倒的にシンプルです。

❷は、実際にGitを使っていると、必要な変更をコミットし忘れたり、逆に不要な変更をコミットしてしまったりするコミットミスが発生します。エディタを使ってコミットを行うと、ソースコードの変更を直接確認しながらコミットが行えるため、このようなミスを限りなく減らすことができます。

ほかにもGitと統合されているAtomでは、

注1）URL https://desktop.github.com/

最新リビジョンからの変更をリアルタイムで視覚化してくれたり、コンフリクトを容易に解決する機能を持っていたりなど、さまざまなメリットを享受できるようになっています。

画面表示と設定

ここでは、Atomの画面の中で操作する、Gitに関係するUIについて解説します。

AtomからGit操作を行うにはコマンドやショートカットキーからでも行えますが、代表的な機能はUIからも行えるようになっているため、慣れていない方でも安心して操作できます。

ステータスバー

AtomでGit管理されているディレクトリを開くと、**図1**のようにステータスバーに現在ブランチ名や変更ファイル数、リモートリポジトリとのコミット差数など、`git status`から得られる情報がリアルタイムで表示されます。

このステータスバーに表示される情報は、すべてクリックすることで、それぞれ対応するGit操作が行えるようになっています（操作は後述します）。

ツリービュー

ツリービューは`git status`と連動し、ファイルの変更を検知して状態に応じた色付けを行ってくれるようになっています（**図2**）。また、それ以外にも設定によって.gitignoreファイルに指定されたファイルや、.gitなど特定のファイルを表示させないようにできます。

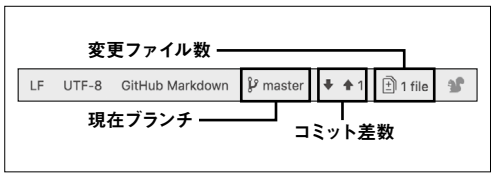
もし.gitignoreに指定されたファイルをツリービューに表示させたくない場合は、ツリービューパッケージの設定から、「Hide VCS Ignored Files」にチェックするか、ツリービューにカーソルがフォーカスしている状態で、“i”[注2]をタイプします。すると、**図2右**のように.gitignoreに指定されているファイルがツリービューに表示されないようになります。

タブ

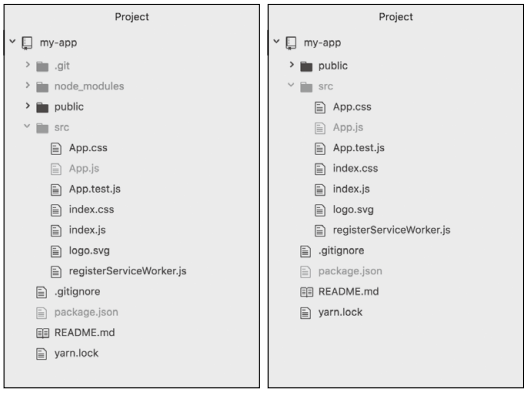
Atomはタブも**git status**と連動させることができます。タブパッケージの設定から「Enable VCS Coloring」にチェックすると、ツリービューと同じようにファイルの変更を検知して色が変化するようになっています（**図3**）。

注2） 表示／非表示を切り替える「Tree View: Toggle VCS Ignored Files」コマンドを実行します。

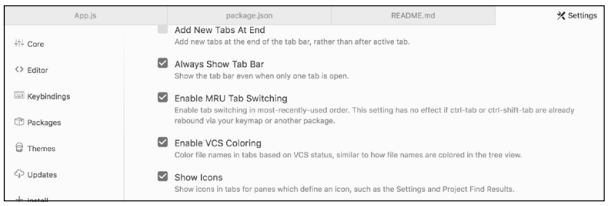
▼図1　ステータスバー



▼図2　ツリービュー（右は.gitignoreファイルを非表示）



▼図3　タブ（白黒でわかりづらいがApp.jsとpackage.jsonはオレンジ色になっている）



Git／GitHubパネル

Git／GitHubパネルはgithubパッケージ[注3]が提供する機能で、その名のとおりGitとGitHubに関する操作が行えます。

Gitパネル（**図4**）では、ファイルのステージ／アンステージ、コミットが行えます。部分的な変更のみをステージするハンクステージにも対応しているため、コミットしたい変更を間違えずにコミットすることができます。

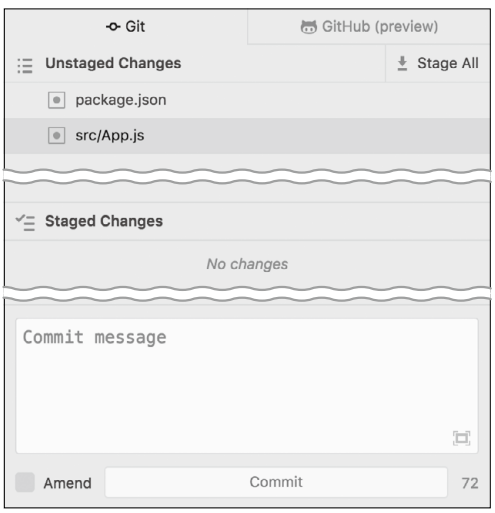
GitHubパネル（**図5**）からは、GitHubに現在ブランチから作成されたプルリクエスト（PR）があれば、その内容を確認できるようになっています。パネル内の「Conversation」をクリックすると、タブが開いてプルリクエストのコメントも確認できます。

AtomによるGit操作

ここからはAtomから行えるGit操作について、おもにGitHubフロー[注4]に必要な操作を例にして解説していきます。

注3）URL https://github.atom.io/
注4）URL https://guides.github.com/introduction/flow/

▼図4　Gitパネル



基本的な操作

まずは、基本的な操作となるブランチ作成からコミットまでの一連の操作をAtomで行ってみましょう。

ブランチを切り替える

まずは、masterブランチから新たなブランチを作成して切り替えます。ブランチの作成／切り替えはステータスバーの現在ブランチ名をクリックして行います。クリックすると、**図6**のようなポップアップメニューが表示され、プルダウンメニューからブランチを切り替えられるようになっています。

新しくブランチを作成したい場合は、「New Branch」ボタンを押します。すると、作成したいブランチ名が入力できるので、こちらにブランチ名を入力して Enter キーを押すと、新しいブランチが作成され、切り替えが行われます。

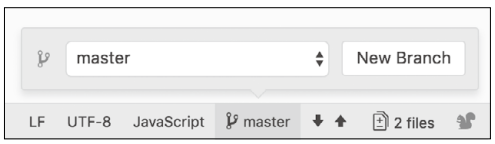
ステージ／アンステージする

次は変更をコミットするためのステージ／アンステージを行います。まず変更をステージするためには、先ほど紹介したGitパネルから行います。Gitパネルを開くには、ステータスバーの変更ファイル数をクリックする、あるいは、GitHub: Toggle Git Tab（Ctrl ＋ Shift ＋ 9）コマンドを実行します。すると、Gitパネ

▼図5　GitHubパネルとConversationボタン



▼図6　ブランチの作成／切り替えメニュー（ステータスバー）



ルが開くので、「Unstaged Changes」に表示されているステージしたいファイル名をクリックします。

ファイル名をクリックすると、**図7**のようなUnstaged Changesタブが開いてファイルの変更個所が表示されます。こちらの右上にある「Stage File」あるいは「Stage Hunk」をクリックすると、変更がステージされます。

また、このUnstaged Changesタブで変更個所をクリックして選択すると、「Stage Hunk」が「Stage Selection」に変化して、選択行のみをステージすることもできます（**図8**）。

アンステージしたい場合は、「Staged Changes」に表示されているファイル名をクリックします。すると、今度はステージされている変更個所が表示されるので、「Unstage File」あるいは「Unstage Hunk」をクリックします。

もし、変更をすべてステージ／アンステージしたい場合は、Gitパネルの「Unstaged Changes」と「Staged Changes」の右に表示される「Stage All」と「Unstage All」をクリックしましょう。

コミットする

コミットは、Gitパネルの最下段にある「Commit message」と薄く表示されているところにコミットメッセージを記入して、その下にある「Commit」ボタンを押すだけです。もし、ボタンを押さずにコミットしたい場合は、Ctrl + Enter、⌘ + Enter、Shift + Enterでコミットを確定できます。

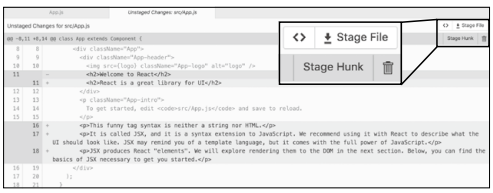
右下にある四角のアイコンをクリックすると、**図9**のように大きな画面でコミットメッセージを記入できます。もし、元に戻したい場合は、同じ場所に表示される「Toggle expanded commit message」をクリックします。

コミットメッセージを書き直す

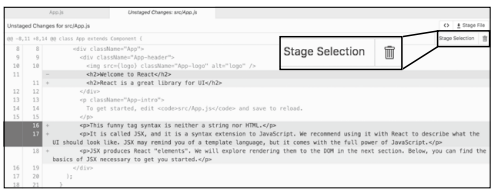
直前に行ったコミットのコミットメッセージを書き直したいときは、「Commit」ボタンの左にある「Amend」チェックボックスにチェックを入れます。すると、**図10**のように直前に行ったコミットメッセージが表示され、コミットログを書き直すことができます。

また、このAmendにチェックを入れたとき、

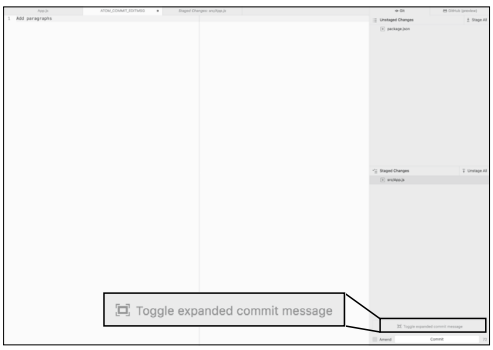
▼図7　Unstaed Changesタブ



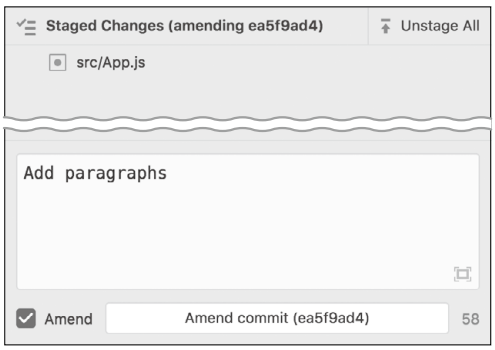
▼図8　行選択をしたUnstaged Changesタブ



▼図9　広くなったコミットメッセージ記入欄



▼図10　Amendにチェックを入れた後



あるいは入れる前にステージを行うと、メッセージを書き直すだけでなく、コミットするファイルも変更可能になります。そのため、ファイルの追加ミスなども修正できるようになっています。

編集中のファイルの変更を取り消す

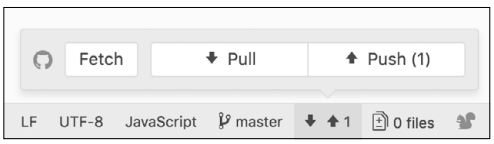
最後は、ファイルの編集を取り消して、HEAD（最新コミット）の状態に戻したいときの方法を紹介します。HEADに戻すには、Editor: Checkout Head Revision（Ctrl + ⌘ + z）コマンドを実行します。

すると、「Confirm Checkout HEAD Revision」というダイアログが表示されるので、OKボタンを押すことでいつでもHEADに戻せるようになっています。なお、もし間違ってHEADへ戻してしまっても、アンドゥで戻す前に戻すことができますので、間違って戻してしまっても変更が失なわれるわけではないので安心して使ってください。

リモートリポジトリに関する操作

次は、リモートリポジトリに関する操作をAtomから行ってみましょう。なお、ここで解説する操作は、プルリクエストの確認以外は、基本的にはGitHubに限ったものではありませんので、ほかのリモートリポジトリに対しても行えます。

▼図11　Fetch／Pull／Pushメニュー



▼表1　fetch／pull／pushに関するコマンド

コマンド名	キーバインド	Gitコマンド
GitHub: Fetch	Alt+g f	git fetch
GitHub: Pull	Alt+g Shift+f	git pull
GitHub: Push	Alt+g p	git push
GitHub: Force Push	Alt+g Shift+p	git push -f

fetch／pull／pushする

リモートリポジトリからfetch／pull、そしてリポジトリへpushするには、ステータスバーのリモートリポジトリとのコミット数の差をクリックします。クリックすると、**図11**のようなポップアップメニューが表示され、「Fetch」ボタン、「Pull」ボタン、「Push」ボタンを押すと、それぞれの操作が実行されます。

fetch／pull／pushは、すべてコマンドからも実行可能になっています。コマンドとショートカットキーを**表1**にまとめたので、参考にしてみてください。なお、ショートカットキーは、Alt + g を押してからいったんキーを離して、次のキーを押す、ツーストロークになっています。

コンフリクトを解決する

マージやリベースの際に、コンフリクトが発生する場合があります。その場合は、コンフリクトを解決しなければマージやリベースを完了できないため、該当するファイルを開いて修正する必要があります。

Atomには、標準でコンフリクトが発生したファイルを修正する機能が備わっています。コンフリクトが発生したファイルをAtomで開くと、**図12**のようにコンフリクトしたコードブロックがハイライトで表示されます。このとき、

▼図12　Atomによるコンフリクト解決



「Use me」ボタンを押して採用したいコードブロックを選択するだけで、自動的にコンフリクトを解決してくれます。

もし、単純な選択だけでなく手修正が必要な場合も、採用したコードブロックのコードを事前に編集できるようになっているので、修正後に「Use me」ボタンを押すことで、手修正を行いつつコンフリクトを解決できます。また、手修正を行うと、**図13**のように「Revert」ボタンが表示され、簡単に修正前の状態に戻すことができるので、安全に編集可能になっています。

GitHubを開く

チーム開発の現場では、Gitだけでなく GitHubもあわせて利用することが多いかと思います。AtomはGitHubが開発したエディタということもあり、GitHubにアクセスするOpen On GitHubという機能が備わっています。

Open On GitHubが提供するコマンドは**表2**になります。`Open On GitHub: Repository`、`Open On GitHub: Issues`、`Open On GitHub: Branch Compare`以外は、編集中のファイルからGitHubページを開くコマンドです。パッケージの設定で行番号をURLに含むかどうか設定可能となっていますので、好みに応じて切り替えてください。



Git関連のAtomパッケージ

最後はGitに関連するAtomパッケージを紹介していきます。Atomは多くのパッケージが公開されています。ここで紹介できるのはその中の一部にすぎませんので、ぜひAtomパッケージのページ[注5]から検索して便利なパッケージを見つけてみてください。



git-plus

これまで解説してきたように、Atom標準機能でもGitHubフローに必要なGit操作が行えます。ですが、マージやリベース、ログの確認などの操作は行えません。もし、これらの操作もAtom上で行いたい場合は、git-plusパッケージをインストールしましょう。

git-plusは**表3**のコマンドなどが利用できます。git-plusはGitに関する操作のほとんどすべてが行えるようになっていますので、Gitをよく利用する人はインストールしておきましょう。



git-blame

`git blame`は指定したファイルに対してコミットの詳細情報を行単位で表示するコマンドです。git-blameパッケージは、編集中のファイルに直接`git blame`の結果を表示します。

注5）URL https://atom.io/packages

▼図13　手修正後のコンフリクト解決



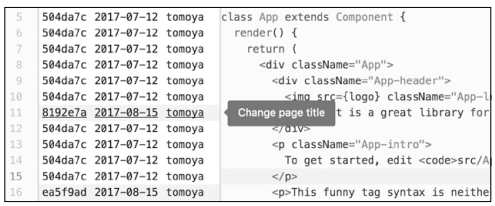
▼表2　Open On GitHubコマンド一覧

コマンド名	キーバインド	説明
Open On GitHub: Repository	Alt+g g	GitHubリポジトリページを開く
Open On GitHub: Issues	Alt+g i	GitHubイシューページを開く
Open On GitHub: File	Alt+g o	編集中のファイルのGitHubページを開く
Open On GitHub: Blame	Alt+g b	編集中のファイルのGitHub Blameページを開く
Open On GitHub: History	Alt+g h	編集中のファイルのGitHub Historyページを開く
Open On GitHub: Copy Url	Alt+g c	編集中のファイルのGitHubページのアドレスをコピーする
Open On GitHub: Branch Compare	Alt+g r	現在ブランチとGitHubのデフォルトブランチの比較ページを開く

▼表3　git-plusの代表的なコマンド

コマンド名	説明
Git Plus: Merge	選択したブランチを現在ブランチにマージする
Git Plus: Rebase	選択したブランチを現在ブランチにリベースする
Git Plus: Log	コミットログ表示する。クリックすると差分を表示する
Git Plus: Log Current File	編集中のファイルのコミットログ表示する
Git Plus: Tags	タグを作成する

▼図14　Git Blame: Toggle実行後



git-blameをインストールすると、Git Blame: Toggle（Ctrl + b）コマンドが利用できます。実行すると、編集中のタブが**図14**のようになり、もう一度実行すると元に戻ります。



git-time-machine

もし、Gitを利用していて過去の状態を確認したくなったときは、git-time-machineパッケージを利用するとよいでしょう。

git-time-machineをインストールすると、Git Time Machine: Toggle（Alt + t）コマンドが利用できます。実行すると、編集中のファイルのコミット履歴をステータスバーの上に表示します。

表示される履歴は、差分行数の量によって円のサイズが大きくなります。円にマウスカーソルをのせるとコミットメッセージがポップアップし、選択すると**図15**のように現在のファイルとの差分を確認できます。

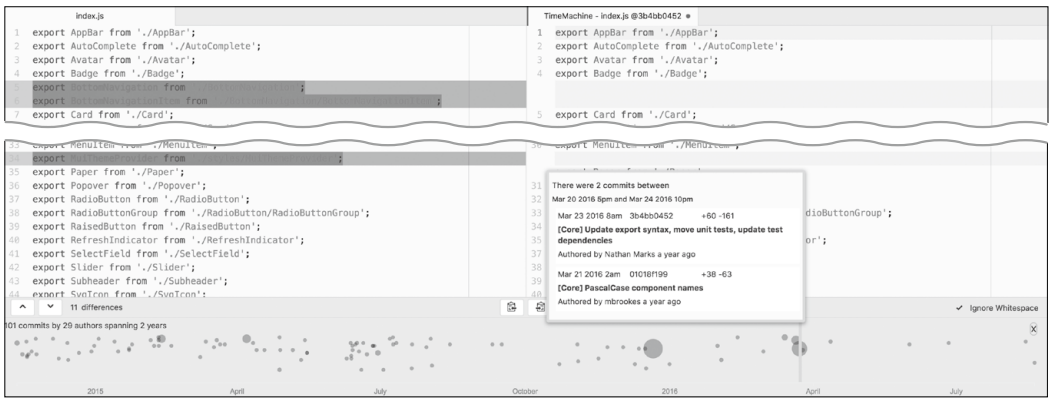


おわりに

ここまでAtomによるGit操作について解説してきましたが、いかがでしたでしょうか？もし、頻繁に利用するコマンドがあれば、キーバインドを登録したり変更すると、より一層スムーズにGitの操作が行えるようになります。

Gitはとても便利で現在の開発になくてはならないツールですが、初心者にとっては少し難しいかもしれません。しかし、AtomからGitを利用すると、徐々に使いこなせるようになっていきますので、初心者の方はぜひ試してみてください。SD

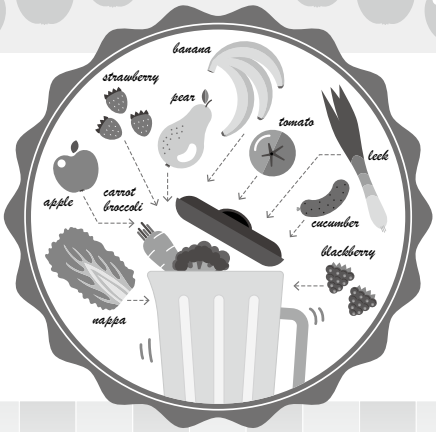
▼図15　Git Time Machineによる差分表示





Git活用の実例［1］

第6章 クラウド＝GitHubとオンプレ＝GitLabを使い分けるピクシブ



第1章で見たように、Gitサービス／プロダクトにはいくつかの選択肢があります。開発現場ではどのように使われているのでしょうか。本章では、pixiv、BOOTH、pixivSketch、Pawooといったサービスを開発・提供しているピクシブ㈱のエンジニアに、その運用スタイルを紹介してもらいます。

AUTHOR 山際 康貴（やまぎわ こうき）
ピクシブ㈱
MAIL yasu@pixiv.com



GitHubとGitLabの特徴とピクシブでの使い方

ピクシブ㈱では、プロダクトのソースコードをGitHub Organizationと自社でホスティングしているGitLabの2つで管理しています。GitLabにはGitLab.comというホスティングサイトも存在しますが、本記事では触れません。本記事では、GitHubとGitLabそれぞれの特徴を紹介しつつ、ピクシブでの運用について説明していきます。



GitHub Organization

GitHub Organizationは、GitHub社が提供している会社などのチームのためのアカウントです。ピクシブが利用しているプランは、旧プランでリポジトリ数による課金体系となっています。現在、リポジトリ数による課金体系は利用できず、契約できるプランはユーザ数による課金体系です。チームの規模に応じてプランがいくつか用意されています[注1]。GitHub Organizationは、GitHubと同じ使い勝手で利用できるため、GitHubに慣れた人にとって扱いが簡単です。ピクシブでは、GitHub Organizationを利用してプロダクトのソースコードを管理しています。

GitHubは、CI（Continuous Integration）やコード評価ツールなどの開発ワークフローを改善するツールの選択肢が豊富です。GitHub Marketplace[注2]ではさまざまなツールが有料／無料で提供されており、開発を高速化したりコード品質を担保する際に役立ちます。また、社外の人とやりとりを行うときには簡単に適切な権限が付与できるため、重宝します。

ピクシブではBOOTH[注3]や、pixivSketch[注4]、Pawoo[注5]などのプロダクトがGitHub Organizationを利用しています。Pawooの開発では、Circle CIとDockerを組み合わせて開発ワークフローの改善を行っています。



GitLab

ピクシブでは、主要サービスであるpixiv[注6]のリポジトリ（pixiv.gitと呼んでいます）をGitLabで管理しています。pixivのコードは昔から社内で管理しており、コードレビュー導入のタイミングでGitLabを使い始めました。

自社ホスティングで運用しており、バージョンアップなどのメンテナンスを自前で行ってい

注1） URL https://github.com/pricing
注2） URL https://github.com/marketplace
注3） URL https://booth.pm/ja
注4） URL https://sketch.pixiv.net/
注5） URL https://pawoo.net/
注6） URL https://www.pixiv.net/

ます。自社ホスティングしている理由は次のとおりです。

【自社ホスティングのメリット】

- 他社のサービスの影響を受けずにコードの修正やデプロイが行える
- 自社のネットワークを利用できるため、「git clone」の実行などが海外でホスティングしている外部サービスよりも高速

自社ホスティングによって、他社のサービスの提供状態に依存せずにアプリケーションコードを修正できます。障害対応時や緊急の修正が必要な場合には、修正がいつでもできる点やネットワークが高速な点は重要なメリットです。

一方で、自社ホスティングには次のようなデメリットも存在します。

【自社ホスティングのデメリット】

- バージョンのアップデートなどのメンテナンスを自社で行わなければならない
- アップデートではGitLabを停止しなければならない（9.1以降のバージョンで無停止で行えるようになった[注7]）

メンテナンスやバージョンアップは自社で行う必要があります。しかし、GitLabはドキュメントが充実しているため、メンテナンスなどで行き詰まる機会は少ないです。とくにバージョンアップの手順はGitLabのリポジトリ[注8]に事細かにまとめてあるため、手順どおり行えばたいていの場合問題は起こりません。一度に複数のバージョンをまたいでアップデートする場合は、必ずすべてのバージョンのアップデート手順を確認してください。多くの場合アップデート手順は同一ですが、ときおり特別な手順が必要な場合があるからです。

また、GitLabはRuby on Railsで書かれているOSSなので、必要に応じてパッチを当ててカスタマイズできることも特徴の1つです。



pixivでのGitLab運用

ピクシブの中で最も大きなリポジトリは、pixiv.gitと呼ばれるソースコード群です。そのリポジトリはGitLab上で管理されており、開発者の多くはGitLabを介してpixiv.gitを毎日編集しています。そのようなpixiv.gitを支えるGitLabの運用とその活用について説明します。



ユーザ認証

pixivの開発サーバでは、LDAPでユーザを管理しています。そのため、GitLabの認証にもLDAPを利用しています。新入社員が入ったときは、LDAPでユーザを作成したあとに、LDAPでGitLabにログインします。ログインするとGitLab上にユーザが作成されるので、GitLabのグループに開発者としてユーザを追加する運用を行っています。

エンジニアならLDAPを使った運用でも問題ありませんが、もしエンジニア以外が頻繁にGitLabを利用するのであればLDAP以外の認証方法のほうがわかりやすいため、お勧めです。GitLabの認証には、LDAP以外にOmniAuthで利用できる多くの認証サービスが使用できます。OmniAuthを使用すれば、GitHubアカウントやGoogleアカウントなどさまざまなアカウントでログインが可能です。アカウントログインの対応範囲については、GitLabのOmniAuth[注9]にまとまっているので参照してください。

GitLabはLDAP、OmniAuthなどの認証方法が多く用意されているので、会社の運用ポリシーごとに適した認証方法が選べます。認証方法の選択肢や詳細については、GitLabの認証と認可[注10]を見てください。それぞれの認証方法を運用するための設定方法も書かれているた

注7）無停止アップデート URL https://docs.gitlab.com/ce/update/README.html#upgrading-without-downtime
注8）URL https://gitlab.com/gitlab-org/gitlab-ce/tree/master/doc/update
注9）URL https://docs.gitlab.com/ce/integration/omniauth.html
注10）URL https://docs.gitlab.com/ce/administration/auth/README.html

め、実際に運用するときに役立ちます。

社外からのセキュアなアクセス

GitLabを運用し始めたころは、ソースコードの漏えいリスクを最小限にするために、ホスティングしているGitLabは社内ネットワークからしかアクセスできないようにしていました。その場合、社外からのアクセスにはVPNを使うしかありません。しかし、緊急対応が必要なときなどにVPN接続をいちいちしていては手間がかかります。

そこで、bitly/oauth2_proxy[注11]を利用してGoogleアカウントによる認証をGitLabの前段に置きました。Google認証を挟むことで、社外からでもセキュアにアクセスできるようになっています。詳しい設定方法や設定の注意点などは、以前公開した当社ブログの記事「GitLabの運用方法をドーンと公開!![注12]」を見てください。

Jenkinsとの連携

GitHubとGitLabのどちらも、さまざまな外部ツールと連携でき、開発効率を向上できます。ピクシブで運用しているGitLabでは、社内で運用しているJenkinsをCIとして連携させています。

連携すると、指定したタイミングであらかじめ指定した操作を実行できます（**図1**）。たとえば、Merge Request（GitHubのPull Requestと同等の機能です）が作成されたタイミングで、自動的にテストを走らせたり、コードの品質を担保するためのLintを動作させたりします。

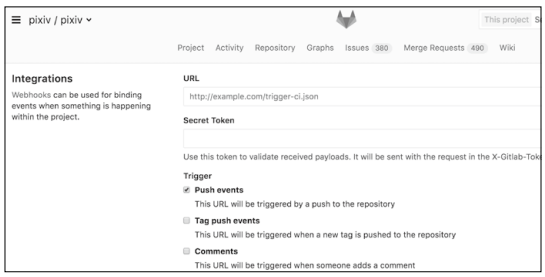
ピクシブでは、Merge Requestが作成されたタイミングでテストとLintを動作させています。実際にテストやLintを実行するのはJenkinsです。連携は、GitLabリポジトリのintegrationsの設定から、Jenkinsへのwebhookを追加して実現します（**図2**）。webhookの設定時に、どの操作がトリガーになるのかを選びます。たとえば、マージされたときや、作成したMerge Requestにコメントが追加されたとき、誰かがPushしたときなどがトリガーとして選べます。

少人数のチームでは、Merge Requestが作成されたときやコメントされたときなど多くのトリガーを設定し、Slackへ通知するように設定しています。連携のおかげでGitLabへアクセスしなくてもチームメイトがどのようなMerge Requestを作成しているのか、どのようなコメントをしているのかわかるようになります。

ピクシブではGitLabとJenkinsを連携して使っていますが、連携できるツールはJenkinsだけではありません。興味のある人は、GitlabのProject integrations[注13]に連携方法や連携できるツールへのリンクがありますので見てください。

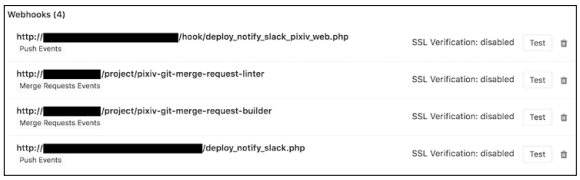
注11）URL https://github.com/bitly/oauth2_proxy
注12）URL http://inside.pixiv.net/entry/2016/12/17/100000
注13）URL https://docs.gitlab.com/ce/user/project/integrations/

▼図1　GitLabの連携画面



プロジェクトの管理画面からPushやMerge Request、コメントなどのイベントをフックとしてほかのアプリケーションを連携できる。設定するときにはプロジェクトのマスター権限かオーナー権限が必要

▼図2　pixivリポジトリのJenkinsとの連携



pixiv.gitでは、masterブランチへのPushをフックとしてSlackに通知したり、Merge Requestの更新時に自動的にテストを実行させている



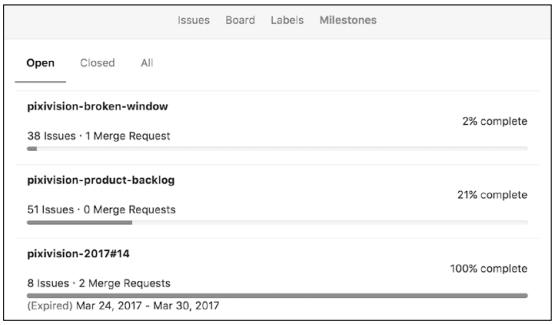
タスク管理ツールとしての利用

GitLabはソースコード管理だけでなく、タスク管理をするための機能がついています。ピクシブのあるチームでは、アジャイル開発のためのタスク管理にGitLabを採用していました。GitLabには、イシューを一覧する機能だけでなく、かんばんのようにタスクの状態を管理するボード、マイルストーンの設定やその進捗を確認できる機能が実装されています（**図3、4**）。ピクシブでは利用していませんが、GitLab Enterprise Editionではプロジェクトのベロシティを知るために便利なバーンダウンチャートが閲覧できます。

ソフトウェア開発のタスク管理ツールとしてGitLabが優れている点は、コードとタスクが一元管理できることです。マイルストーンを設定しておけば、イシューに担当者をアサインしたり、クローズしたりすると、マイルストーンに含まれるタスクのステータスが自動的に変更されます。GitLabや他のタスク管理ツールの間を行き来する必要はありません。タスク管理ツールとしての運用方法やツールの印象は、ピクシブのブログ記事を参照してください[注14]。

ほかにも進捗状況を管理するための機能が、GitLabには備わっています。たとえば、Merge Requestやイシューに対して見積もりと実際にかかった時間を記録するためのタイムトラッキング機能がバージョン8.14からβ版として実装されています。ピクシブではこれを使った運用をしたことはありませんが、開発者のタスク消化時間などを集計し見直す動きがあるため、今後開発者の生産性向上に役立つかもしれないと考えています。

▼図3　GitLabのマイルストーン画面



プロジェクトなどの単位でマイルストーンを作成し、タスクをイシューとして登録して利用する。この画面では、複数あるマイルストーンの進捗状況がそれぞれ確認できる。ひとつひとつのマイルストーンは、オーナーあるいはマスター権限のユーザが作成できる



まとめ

本記事では、ピクシブでのGitHubとGitLabの使い分けや実際のGitLab運用について紹介しました。GitHubやGitLabはどちらも精力的に開発されています。今後も新機能が追加されて、開発速度をさらに加速させてくれるだろうと期待しています。今回はピクシブでの運用事例を紹介しましたが、GitHubとGitLabどちらについても一部の機能しか使っていないはずです。公式ドキュメントなどを参照してどのようなことができるか、ぜひ調べてみてください。SD

注14）「アジャイル開発のタスク管理にGitLabを使う理由 ― GitLab meetup #01 レポート」URL https://inside.pixiv.blog/uchien/828

▼図4　GitLabのボード画面

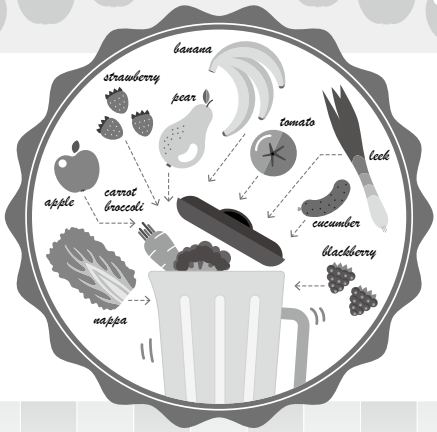


それぞれのイシューがどのフェーズにいるのか一覧できる機能。開発中のものはどのイシューなのか、デプロイ直前のものがどれなのか確認できる。また、ドラッグ＆ドロップでフェーズを簡単に変更できる



Git活用の実例［2］

第7章 GitLab CI＋RedPenでドキュメント検査もCIしよう！



本章ではGit活用事例として、GitLabに統合されているCI（継続的インテグレーション）ツールであるGitLab CIと、Java製のドキュメント自動検査ツールRedPenを連携して、書籍執筆に活用する方法や、Javaを使用しないプロジェクトでRedPenを実行する方法を紹介します。

AUTHOR 打田 智子（うちだ ともこ）
㈱ニューズピックス



RedPenとは

RedPenは、伊藤敬彦氏（@takahi_i）により開発が進められている、Java製のドキュメント検査（校正）ツールです。「校正ルールを設定ファイルで柔軟に設定できる」「言語非依存（多言語に対応）」「MarkdownやAsciiDocなど、多様なマークアップ言語をサポート」といった特徴があります。

RedPenのインストールや動作方法については、次の公式サイト、ブログ、書籍を参照してください。

・公式サイト：URL http://redpen.cc/
・公式ブログ：URL https://blog.redpen.cc/
・書籍『ドキュメント作成システム構築ガイド』：URL http://gihyo.jp/book/2016/978-4-7741-8036-6



GitLab CIからRedPenを実行する

それではさっそく、GitLabプロジェクトで管理しているドキュメントを、GitLab CIとRedPenを使って継続検査する方法を見ていきましょう。

GitLab CIは、GitLabに統合されているCI（Continuous Integretion：継続的インテグレーション）ツールです。GitLabで管理しているプロジェクトであれば、設定を追加するだけで使うことができます。

GitLab CIを使うためには、リポジトリのルート直下に.gitlab-ci.ymlというYAML設定ファイルを用意します。また、RedPenを動作させるためにJava 8がインストールされたRunner（ジョブを実行する仮想環境）が必要です。

GitLab CIの設定方法や、Runnerについて詳しくは次のドキュメントを参照してください。

・Getting started with GitLab CI：
URL https://docs.gitlab.com/ce/ci/quick_start/README.html
・Runners：
URL https://docs.gitlab.com/ce/ci/runners/README.html



GitLab CIの設定

たとえば、プロジェクトルート以下のdocs/sample.mdというMarkdownフォーマットのファイルを検査する.gitlab-ci.ymlは**リスト1**のようになります。なお校正ルールはredpen/redpen-conf-ja.xmlに記載されているとします。

リスト1を最初から順番に見ていきましょう。

①で、Runnerのイメージを指定しています。前述のようにRedPenの動作にはJava 8以上が必要ですので、`image: java:8`と指定します。imageは、DockerイメージからRunnerを起動するキーワードです。Docker Hub[注1]にホストされている任意のイメージを指定できます。

②では、ビルド環境の環境変数にRedPenのバージョンを設定しています。異なるバージョンのRedPenを使う場合は、REDPEN_VERSIONを変更してください。

③のbefore_scriptには事前処理を行うスクリプトを記述します。ここではRedPenのリリースモジュールをジョブ実行環境にダウンロードして展開しています。

④はメインとなる処理です。redpenコマンドを実行し、結果を標準出力に出力しています。-cオプションにRedPen設定ファイルを指定します。-fオプションにはファイルフォーマットを指定します。なお、拡張子からMarkdownであることを推定して動作するため、ここでは-fオプションは必須ではありません。

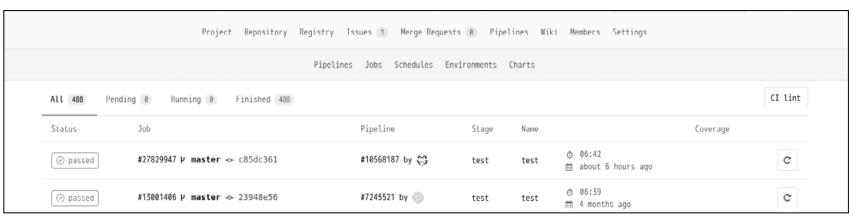
作成した.gitlab-ci.ymlをGitLabにpushすると、pushをトリガとしてジョブが実行されます。RedPenの出力（検査結果）は、ジョブのログより確認できます。ジョブのログを表示するには、メニューの「Pipelines > Jobs」より、ジョブのStatusアイコンをクリックしてください（**図1**）。

注1）URL https://hub.docker.com/

▼リスト1　Markdownフォーマットのファイルを検査する「.gitlab-ci.yml」

```
image: java:8  ……①
variables:     ……②
  REDPEN_VERSION: "1.9.0"
before_script:   ……③
  - wget -q https://github.com/redpen-cc/redpen/releases/download/redpen-${REDPEN_VERSION}/redpen-${REDPEN_VERSION}.tar.gz
  - tar xzf redpen-${REDPEN_VERSION}.tar.gz
test:   ……④
  script:
    - redpen-distribution-1.7.4/bin/redpen -c redpen/redpen-conf-ja.xml -f markdown docs/sample.md
```

▼図1　ジョブのログを確認する



実際のプロジェクトでの活用例

活用例1：書籍執筆

先ほどの例では、1つのファイルのみを検査していましたが、実際のプロジェクトでは多数のファイルがあり、かつ階層化されている場合も多いと思います。本項では、実際のプロジェクト例として、筆者が共著で関わった書籍『[改定第3版] Apache Solr入門』[注2]の執筆における活用例を紹介します。

この書籍は、GitLabのプライベートプロジェクトを使って複数人で執筆し、すべての原稿を1つのプロジェクトで管理していました。原稿管理プロジェクトのディレクトリ構成を**図2**に示します（一部省略）。検査対象の原稿は、chXXというディレクトリに、章ごとに管理されています。

注2）打田智子、大須賀稔、大杉直也、西潟一生、西本順平、平賀一昭 著、㈱ロンウイット、㈱リクルートテクノロジーズ 監修、『[改訂第3版] Apache Solr入門』、技術評論社、2017年4月、ISBN978-4-7741-8930-7。

このプロジェクトのドキュメント検査を行う.gitlab-ci.yml（抜粋）を**リスト2**に示します。1つのファイルのみ検査したときとの違いは、各章ごとにディレクトリに含まれるファイルを、xargsを使ってまとめて検査している点です。scriptには任意のコマンドがいくつでも書けるので、ご自身のプロジェクトに合わせて柔軟に変更してください。

▼図2　原稿管理プロジェクトのディレクトリ構成

```
manuscripts/
├─ .git
├─ .gitignore
├─.gitlab-ci.yml
├─ README.md
├─ ch00
├─  ch01
│   ├─ 1.0.md
│   └─ 1.1.md
│   ⋮
├─ ch02
├─ ch03
└─ ch04
⋮
├─ redpen
│   ├─ logback.xml
│   └─ redpen-conf-ja.xml
⋮
```

リスト2のCIを回したときの出力結果（抜粋）を**図3**に示します。助詞「の」の重複や1文の長さが長過ぎるなど、さまざまな指摘が報告されていることがわかります。ドキュメントの特性によって、無視しても問題ない報告もあるので、対応するかどうかは文脈から判断すると良いでしょう。



活用例2：non-Javaプロジェクトでの利用

ここまで、image: java:8を使用し、Java 8があらかじめインストールされたRunner上でジョブを実行してきました。しかし、Javaを使用しないプロジェクトでRedPenを実行したくなることもあるでしょう。本項では、non-Javaプロジェクト向けのRunnerでRedPenを実行する設定を紹介します。

リスト3は、PythonプロジェクトのGitLab CIジョブの設定例です。docs/ディレクトリ以下に、プロジェクトに関連するドキュメントが同梱されているものとします。

最初から順番に見ていきましょう。

①ではPython 3.6用のイメージを指定しています。

▼リスト2　プロジェクトのドキュメント検査を行う「.gitlab-ci.yml」

```
image: java:8
variables:
  REDPEN_VERSION: "1.9.0"
before_script:
  - wget -q https://github.com/redpen-cc/redpen/releases/download/redpen-${REDPEN_VERSION}/redpen-${REDPEN_VERSION}.tar.gz
  - tar xzf redpen-${REDPEN_VERSION}.tar.gz
test:
  script:
    - cp redpen/logback.xml redpen-distribution-${REDPEN_VERSION}/conf/logback.xml
    - ls ch01/*.md | xargs -I {} redpen-distribution-${REDPEN_VERSION}/bin/redpen -c redpen/redpen-conf-ja.xml -f markdown {} || true
    - ls ch02/*.md | xargs -I {} redpen-distribution-${REDPEN_VERSION}/bin/redpen -c redpen/redpen-conf-ja.xml -f markdown {} || true
    - ls ch03/*.md | xargs -I {} redpen-distribution-${REDPEN_VERSION}/bin/redpen -c redpen/redpen-conf-ja.xml -f markdown {} || true
    - ls ch04/*.md | xargs -I {} redpen-distribution-${REDPEN_VERSION}/bin/redpen -c redpen/redpen-conf-ja.xml -f markdown {} || true
    ...
    - ls ch_appendix/*.md | xargs -I {} redpen-distribution-${REDPEN_VERSION}/bin/redpen -c redpen/redpen-conf-ja.xml -f markdown {} || true
```

▼図3　リスト2の出力結果（抜粋）

```
(..略..)
$ ls ch10/*.md | xargs -I {} redpen-distribution-${REDPEN_VERSION}/bin/redpen -c redpen/
redpen-conf-ja.xml -f markdown {} || true

[2017-08-04 04:23:05.919][ERROR] cc.redpen.Main - The number of errors "6" is larger than
specified (limit is "1").
10.2_モニタリング.md:46: ValidationError[JapaneseStyle], Found invalid Japanese Style
"である" at line: Solrは，Java におけるリソースのモニタリングや監視，操作を行うための標準 API
である JMX（Java Management Extensions）に対応しています。
10.2_モニタリング.md:246: ValidationError[CommaNumber], The number of commas (6) exceeds
the maximum of 3. at line: その他，サーバー筐体の温度や，CPU，メモリ，ディスク I/O
などのマシンリソースといった，システム運用時に監視すべき一般的な項目ついても，確認しておきましょう。
```

▼リスト3　PythonプロジェクトのGitLab CIジョブの設定例

```
image: python:3.6 ……①
variables:        ……②
  REDPEN_VERSION: "1.9.0"
before_script:    ……③
  - echo deb http://http.debian.net/debian jessie-backports main >> /etc/apt/sources.list
  - apt-get update
  - apt-get install -t jessie-backports ca-certificates-java -y
  - apt-get install openjdk-8-jdk -y
  - update-java-alternatives -s java-1.8.0-openjdk-amd64
test:   ……④
  script:
    - python setup.py test
after_script:  ……⑤
  - wget -q https://github.com/redpen-cc/redpen/releases/download/redpen-${REDPEN_VERSION}/
redpen-${REDPEN_VERSION}.tar.gz
  - tar xzf redpen-${REDPEN_VERSION}.tar.gz
  - redpen-distribution-${REDPEN_VERSION}/bin/redpen -c docs/redpen/redpen-conf-ja.xml -f
rest docs/index.rst
```

②では使用するRedPenのバージョンを環境変数に格納しています。

③で、OpenJDK 8をインストール[注3]し、update-java-alternativesコマンドでデフォルトJavaのパスを変更しています。

④ではCI対象であるPythonプロジェクト本体のテストを行います。

⑤のafter_scriptには事後処理を行うスクリプトを記述します。ここで、同梱ドキュメントdocs/index.rstの検査を行います。拡張子.rstのファイルはreSTフォーマットのファイルです[注4]。

注3）JDKインストールの前にaptパッケージ一覧を更新しています。DockerイメージのUbuntuバージョンが古く、デフォルトではOpenJDK 8のパッケージが見つからないためです。

注4）reST（reStructuredText）とは、Docutils（URL http://docutils.sourceforge.net/）というPython製のテキスト処理フレームワークが定める軽量マークアップ言語です。



まとめ

本章では、GitLab CIとRedPenを連携し、GitLab管理のドキュメントを自動検査する方法を紹介しました。ソフトウェア開発をしていると、付随するユーザマニュアルなどのドキュメントを書く機会も多いと思います。ソースコードをユニットテストや静的解析ツールで検査するように、ぜひドキュメントも継続検査していきましょう！SD

BACKNUMBER バックナンバーのお知らせ

バックナンバー好評発売中！常備取り扱い店もしくはWebより購入いただけます

本誌バックナンバー（紙版）はお近くの書店に注文されるか、下記のバックナンバー常備取り扱い店にてお求めいただけます。また、「Fujisan.co.jp」（http://www.fujisan.co.jp/sd）や、e-hon（http://www.e-hon.ne.jp）にて、Webから注文し、ご自宅やお近くの書店へお届けするサービスもございます。



2017年9月号

第1特集
Web技術【超】入門
いま一度振り返るWebのしくみと開発方法

第2特集
tmux&Byobu
開発効率アップのターミナル改造術

一般記事
・認証を支える技術
・Ejectコマンドで遊んでみませんか?

定価（本体1,220円+税）



2017年8月号

第1特集
私も機械学習エンジニアになりたい!
先端Web企業の取り組み方は?

第2特集
エンジニアのための うけるプレゼン・すべるプレゼン
——あなたの思いを伝える技術、教えます

一般記事
・「Mastodon」旋風からわかるSNSの未来

定価（本体1,220円+税）



2017年7月号

第1特集
もっとbashを使いこなしませんか?
理論&応用でシェル力の幅を広げる

第2特集
データの抽出・加工に強くなる!
MySQL［SELECT文］集中講座

一般記事
・ハッシュ関数を使いこなしていますか?（後編）
・Jamesのセキュリティレッスン【10】
・Windows Server 2016で構築する最新ファイサーバ（後編）

定価（本体1,220円+税）



2017年6月号

第1特集
Vim、Emacs、Atom、Visual Studio Code
あなたのプログラミングを加速させるエディタ

第2特集
多用途に使いこなせ、コードが読みやすく保守しやすい
今すぐはじめるPython

一般記事
・ハッシュ関数を使いこなしていますか?（前編）
・Windows Server 2016で構築する最新ファイサーバ（前編）

定価（本体1,220円+税）



2017年5月号

第1特集
先輩が教える㊟ノウハウ
Linux入門【UNIXネットワーク編】

第2特集
サイボウズ流　サービス改善につなげる
ドッグフーディング環境の作り方

第3特集
いまから学ぶブロックチェーンのしくみ

定価（本体1,220円+税）



2017年4月号

第1特集
基礎から学べば役に立つ!
目的から考える! 実作業から学ぶLinux入門（OS操作編）

第2特集
AWS LambdaとMicrosoft Azure Functionsで体験
はじめてのサーバーレスアーキテクチャ

一般記事
・mBaaSのしくみ紹介

定価（本体1,220円+税）

Software Design　バックナンバー常備取り扱い店							
北海道	札幌市中央区	MARUZEN & ジュンク堂書店　札幌店	011-223-1911	神奈川県	川崎市高津区	文教堂書店　溝の口本店	044-812-0063
	札幌市中央区	紀伊國屋書店　札幌本店	011-231-2131	静岡県	静岡市葵区	戸田書店　静岡本店	054-205-6111
東京都	豊島区	ジュンク堂書店　池袋本店	03-5956-6111	愛知県	名古屋市中区	三洋堂書店　上前津店	052-251-8334
	新宿区	紀伊國屋書店　新宿本店	03-3354-0131	大阪府	大阪市北区	ジュンク堂書店　大阪本店	06-4799-1090
	千代田区	書泉ブックタワー	03-5296-0051	兵庫県	神戸市中央区	ジュンク堂書店　三宮店	078-392-1001
	千代田区	丸善　丸の内本店	03-5288-8881	広島県	広島市南区	ジュンク堂書店　広島駅前店	082-568-3000
	中央区	八重洲ブックセンター本店	03-3281-1811		広島市中区	丸善　広島店	082-504-6210
	渋谷区	MARUZEN & ジュンク堂書店　渋谷店	03-5456-2111	福岡県	福岡市中央区	ジュンク堂書店　福岡店	092-738-3322

※店舗によってバックナンバー取り扱い期間などが異なります。在庫などは各書店にご確認ください。

DIGITAL デジタル版のお知らせ

デジタル版Software Design 好評発売中！紙版で在庫切れのバックナンバーも購入できます

本誌デジタル版は「Fujisan.co.jp」（http://www.fujisan.co.jp/sd）、「雑誌オンライン.com」（http://www.zasshi-online.com/）、「Gihyo Digital Publishing」（https://gihyo.jp/dp）で購入できます。最新号、バックナンバーとも1冊のみの購入はもちろん、定期購読も対応。1年間定期購読なら5%割引になります。デジタル版はPCのほかにiPad／iPhoneにも対応し、購入するとどちらでも追加料金を払うことなく、読むことができます。



第2特集

脆弱性スキャナ Vuls入門

システムのセキュリティチェックをもっと楽に

脆弱性発表 → セキュリティパッチが公開 → 自社システムへの影響調査 →
必要に応じて適用 → また脆弱性発表……。
必要なこととはいえ、この反復作業はシステム管理者のモチベーションをダダ下がりさせます。
その辛い作業を改善すべくリリースされたのが、脆弱性スキャナ「Vuls」です!
Vulsはシステムの脆弱性の有無を自動でチェックしてくれます。
Linuxサーバはもちろん、FreeBSDも、コンテナも。
さあ、Vulsを導入して、新たな脆弱性にも動揺しない平穏なシステム運用を実現しましょう。



第1章 P.70
Author 林 優二郎
セキュリティ対応に追われる現場の悲鳴と、その解決策



第2章 P.75
Author 枇榔 晃裕、牛田 隆之
Vuls導入チュートリアル



第3章 P.83
Author 牛田 隆之
Vulsとさまざまなプロダクトとの外部連携



第4章 P.88
Author 神戸 康多
Vuls誕生秘話 ――OSSをバズらせるには



第5章 P.91
Author 福田 鉄平
実際の運用と、今後のVulsの進化

イラスト：高野 涼香

第1章 セキュリティ対応に追われる現場の悲鳴と、その解決策



Author 林 優二郎（はやし ゆうじろう）・フューチャーアーキテクト㈱
Twitter @vuls_ja

大規模なセキュリティ事件が相次ぐ昨今、OS・ソフトウェアの脆弱性への迅速な対応が叫ばれています。そんな中、生まれたVulsは、脆弱性検知を自動化してくれるツールです。本章ではVulsの具体的な設定・操作に入る前に、Vulsができること、できないことを押さえましょう。



セキュリティリスクの高まりと現場の課題

高まるセキュリティ対策の必要性

日本年金機構の情報漏洩事件を1つのきっかけとして、日本でもサイバーセキュリティの重要性が認識されるようになり、国家を挙げて、サイバーセキュリティに対して危機感を持って取り組むようになりました。企業や団体でも、急速にサイバーセキュリティ対策が進んだように思います。しかし、いまだに情報漏洩などのセキュリティインシデントは世間を騒がせています。対策が進んだはずの日本企業で、なぜセキュリティインシデントは減らないのでしょうか。

現場の課題

日本年金機構を始めとしたセキュリティインシデント発生の頻度が増え、国がガイドラインなどを整備するのに合わせ、企業でもさまざまなサイバーセキュリティ対策が進められました。ファイアウォールやIPS/IDS（不正侵入検知・防御システム）、WAF（Webアプリケーションファイアウォール）、SIEM（Security Information and Event Management）などのさまざまなセキュリティ製品が導入され、企業のサイバーセキュリティ対策は強固になりました。

しかし、それでもセキュリティインシデントは増え続けています。攻撃が巧妙化し、セキュリティ対策製品の目をかいくぐるようになったというのも原因の1つですが、もう1つの原因として、セキュリティパッチ（以下、パッチ）が公開されてから攻撃までの期間が短くなっているということが挙げられます。パッチが公開されるとそのソフトウェアに含まれる過去バージョンの脆弱性情報が公開されるため、攻撃者からすると攻撃の手立てを講じやすくなります。そのため、情報公開された脆弱性に対して早急にパッチを適用しないと、攻撃のリスクは未知の脆弱性よりも高くなります。IPS/IDSやWAFなどのセキュリティ製品が防いでくれる可能性もありますが、パッチが公開されてからセキュリティ製品にシグネチャが反映されるまでには時間を要するため、その間の攻撃を防ぐことができません。早急にパッチを適用することは、非常に重要なセキュリティ対策なのです。

今年猛威を振るったランサムウェア「WannaCry」のケースでも、SMB（Server Message Block）の脆弱性を突いて不正アクセスが行われましたが、すでに公開されているパッチを迅速に適用していれば、被害を未然に防げたのです。

一方でパッチ適用には課題もあります。年間9,000件以上もの脆弱性が発見されている現在、それに対応するパッチの数もまた膨大となりま

す。それらの情報を毎日人力で調査し続けるのはたいへんです。しかも、大半が自分のシステムに関係ない脆弱性の情報です。それだけの数の脆弱性を人が調査する場合には、抜け漏れが発生する恐れがあります。調査や対応が一度漏れた脆弱性は長期間放置されることが多く、十分なセキュリティレベルを保てるとは言えません。また、セキュリティに対してそれほど大きな投資をする企業もまだまだ少ないため、ほかの業務と兼任でセキュリティ担当者が決められることも多く、本来注力すべきセキュリティ業務に時間を割けないことも多いです。

これらのセキュリティ運用の現場の課題を解決すべく開発されたソフトウェアが「Vuls（VULnerability Scaner）」です。



Vulsの機能概要

世界中で報告された脆弱性を管理しているデータベースとしてJVN[注1]、NVD[注2]があります。

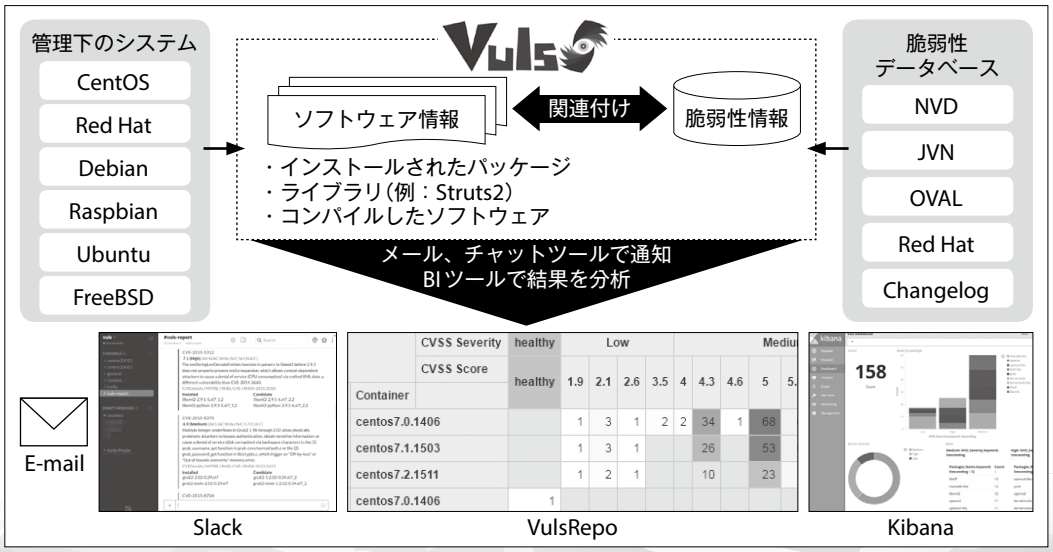
Vulsは管理下にあるシステムのソフトウェア情報を収集し、これら脆弱性データベースの情報と関連付け、管理下のシステムに関係のある脆弱性情報のみを通知してくれます（**図1**）。米国政府が推進している、情報セキュリティに関わる技術面での自動化と標準化を実現する技術の1つであるOVAL[注3]のリポジトリも一部参照しており、今後はOVAL以外にも参照する脆弱性情報を増やすことで、検知精度の向上と検知範囲の拡大を進める予定です。

検知対象はOSのパッケージだけでなくミドルウェアなども含みます。対象プラットフォームとしては、主要なLinuxディストリビューション、FreeBSD、Dockerコンテナに対応しています。通知は、メールはもちろんSlackなど、さまざまな方法が可能です。

Webアプリケーションを操作することで安全性を確認するペネトレーションテストとは異なり、VulsはOSパッケージやライブラリなどに含まれる脆弱性を検知することで安全性の確認を行います。Vulsを定期実行することで、脆弱性調査・対応の抜け漏れを防止し、スピーディーに対応可能になります。その結果、セキュリティ

注1）Japan Vulnerability Notes URL https://jvn.jp/
注2）National Vulnerability Database URL https://nvd.nist.gov/
注3）Open Vulnerability and Assessment Language URL http://oval.mitre.org/

▼図1　Vulsのしくみ概要



リスクの低減につながるとともに、膨大な数の脆弱性の調査に力を割く必要がなくなり、本来人がやるべき業務に注力できるようになります。

Vulsができないこと

Vulsが実施するのは脆弱性の検知までです。自動でのアップデートは実施しません。これは、Vulsが自動でアップデートまで行うと、サーバで動作しているアプリケーションが動作しなくなる可能性があるため、必ず人による判断が必要であると考えているためです。

Vulsの特長

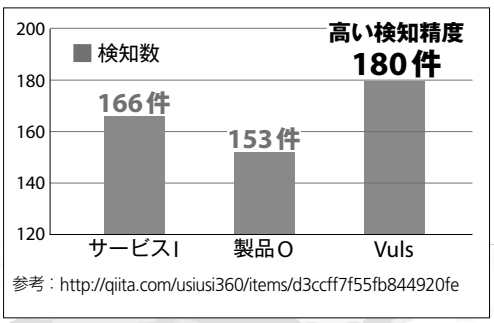
Vulsのおもな特長は次の5つになります。

①高い検知精度
②検知範囲の広さ
③エージェントレス
④セットアップ、初期設定が簡単
⑤豊富なレポート手段

①高い検知精度

同様の機能を持ったほかのソフトウェア、サービスと比べても高い検知精度を誇ります。**図2**はAmazon Linuxに対して脆弱性スキャンを実行した場合の検知数の比較結果です。Vulsと類似の脆弱性検知サービスやツールと比べても、1～2割の検知数の差が出ています。これはVulsが、JVNやNVDだけでなくOVALの情報も組み合わせることで検知精度を上げているためです。

▼図2　脆弱性検知数の比較



参考：http://qiita.com/usiusi360/items/d3ccff7f55fb844920fe

今後はこの検知精度にさらなる磨きをかけていき、検知精度でほかの追随を許さないソフトウェアへと成長させていく予定です。

②検知範囲の広さ

さまざまな検知対象に対応しています。主要なLinuxディストリビューションはもちろん、FreeBSDやDockerなどのコンテナ、IoTで使われるRaspbianなどのOSにも対応しています。ネットワーク機器についても、CPE[注4]情報を設定ファイルに登録することで検知が可能です。同様にCPE情報を登録することで、OSパッケージだけでなく、プログラミング言語のライブラリ、フレームワーク、ミドルウェアなども検知できます。

登録が必要なソフトウェアが多い場合には、OWASP Dependency Check[注5]により出力されるXMLファイルを設定することで、CPE情報の自動登録も可能です。

③エージェントレス

SSHで各サーバに接続してスキャンを行うため、サーバごとにモジュールをインストールする必要がありません。複数のサーバからなる既存環境においても、サーバ1つにモジュールをインストールするだけで容易に導入できます。

動作としてはモジュールをインストールしたサーバからSSHで各サーバに接続し、いくつかのコマンドを実行するだけですので、サーバに負荷はほとんどかかりません。システム情報を変更するような動作はせず、環境非破壊のスキャンのため、クラウドサービスやデータセンター上のシステムに導入する場合にも、ペネトレーションテスト実施時のような申請は必要ありません。

④インストール、初期設定が簡単

Go言語で開発されているため、クロスコンパイルに対応しており、主要なプラットフォームで動作します。前述のとおり、モジュールを配置し、

注4）Common Platform Enumeration。詳細は第2章で解説。
注5）脆弱性検知ツールの1つ。詳細は第2章で解説。

環境に合わせて設定ファイルを変更するだけで動作するので、面倒な初期設定が必要ありません。

⑤豊富なレポート手段

メールやSlack、ファイル出力やTUI(Terminal-Based User Interface)など、さまざまなレポート手段に対応しています。セキュリティ情報には英語の情報源が多いですが、日本語の情報があるものについては、日本語で表示できます。

構成パターン

Vulsはエージェントレスのスキャナのため、スキャン対象のサーバに対してSSHで接続できるサーバにインストールする必要があります。本番稼働している全サーバをスキャンする必要があるとすると、踏み台サーバ[注6]にインストールするのが最も適していると考えられます。サーバ台数が数百台程度であれば、Vulsの動作にはリソースをほとんど割かなくて良いため、踏み台サーバにインストールしたとしても、本番運用にはほとんど影響を与えることなく動作可能です。

インストールができたら、次の接続環境の確認を行い、必要な設定変更を行います。

・スキャン対象の各サーバからパッケージのリポジトリへの接続
・インストールサーバ(踏み台サーバ)からJVN、NVDなどのデータベースへの接続

ファイアウォールなどで接続を制限している場合、ファイアウォールの接続許可設定が必要になります。設定が完了したら、あとはスキャンを実施するタイミング(半日ごと、日次など)を決め、cronなどのスケジューラで定期実行するように設定すれば、セットアップは完了です。

導入により見込める効果

Vuls導入により次の5つの効果が見込めます。

①セキュリティリスク低減
②作業の効率化、定型化
③調査・対応漏れ防止
④属人化排除
⑤対応スピード向上

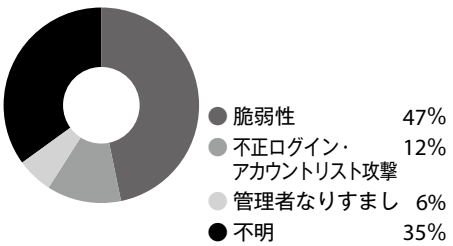
①セキュリティリスク低減

「現場の課題」の節で述べたとおり、ほとんどのセキュリティ攻撃は脆弱性を突いた操作が必要であり、脆弱性をなくすことがセキュリティリスクの低減につながります。

脆弱性への対応がセキュリティリスクを低減することが他社の調査結果(**図3**)からも見て取れます。Trend Microの調査によると、公開さ

注6) DMZなどに配置される、外部との通信の際に経由する必要のあるサーバ。SSHゲートウェイ、SSHプロキシとも呼ばれる。

▼図3 他社リサーチ結果



2016年上半期に公表・報道された公開サーバへの攻撃による情報漏えい事例の攻撃手法割合

「2016年上半期 セキュリティラウンドアップ」
https://www.trendmicro.co.jp/cloud-content/jp/pdfs/security-intelligence/threat-report/pdf-sr2016h1-20160823_01.pdf



「Google Japan Blog——サイバーセキュリティ月間：エキスパートたちのセキュリティ対策」
http://googlejapan.blogspot.jp/2016/02/blog-post.html

れているサーバへの攻撃の47%が脆弱性を突いた攻撃であるという結果が出ています。また、Google Japanの調査結果では、エキスパートが行う対策は「ソフトウェア・アップデートをインストールする」となっています。どちらの結果もソフトウェア・アップデートを行い、既知の脆弱性をなくすことがセキュリティリスク低減に効果的であることを示しています。

②作業効率化、定型化

年間9,000件以上にもおよぶ脆弱性の情報収集を自動化してくれます(**図4**)。この膨大な量の調査を人が行うのは現実的ではなく、実施する人のモチベーション低下を招きます。自動化することで、本来人が力を割くべき対策の検討やパッチ検証などの作業にフォーカスできるようになります。

③調査・対応漏れ防止

脆弱性の調査・対応を人が行う場合、調査時や対応時に漏れが発生すると、そのあと再び調査されることはほとんどないため放置されたままとなります。これらは年月を経るごとに蓄積され、多くの脆弱性が未対応のままになってしまいます。Vulsはシステマティックに検知を行うため検知漏れの心配はなく、対応漏れが発生した場合も次のスキャン時に検知することで、対応漏れを防止することができます(**図5**)。

④属人化排除

脆弱性の調査方法や対象サーバの洗い出しの精度は、実施する人のスキルやノウハウに依存するところが多く、内容にばらつきが生じます。Vulsを使えば属人化を排除し、セキュリティレベルを一定に保てます。新メンバー参画時にも、教育・スキルアップが必要な範囲が少なくなるため、柔軟なチーム体制の構築が可能です。

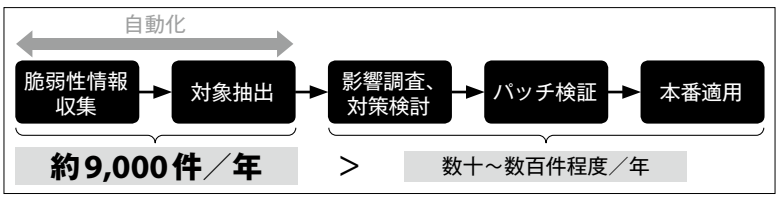
⑤対応スピード向上

脆弱性の検知を人が行う場合には、パッチ公開から数日～数週間程度かかることも少なくないですが、Vulsを使えば即時検知が可能になります。脆弱性が出た日に検知して通知してくれるため、すぐに対応に移れます。近年のサイバー攻撃はパッチ公開直後に行われることが多く、迅速に対応できることが非常に重要です。

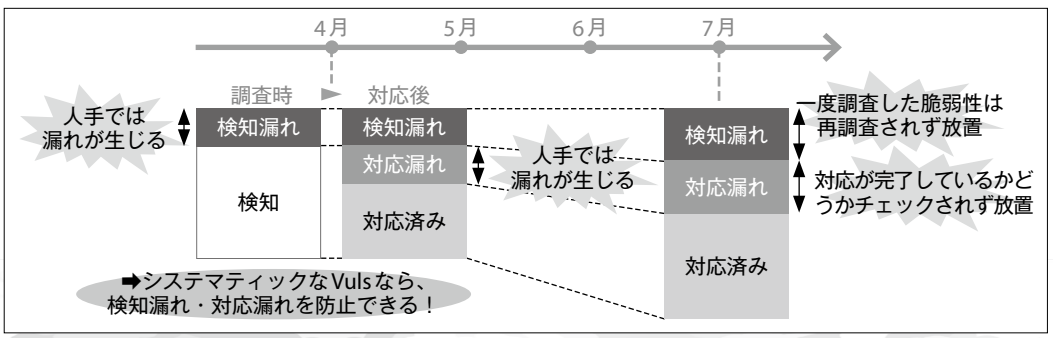
◆ ◆ ◆

以上、脆弱性対応の必要性とVulsの役割、特長について概要を述べました。次章以降では具体的なVulsの導入方法を説明します。 SD

▼図4　効率化の効果



▼図5　人手における対応漏れ・検知漏れ



第2章 Vuls導入チュートリアル



Author 枇榔 晃裕（びろう あきひろ） フューチャーアーキテクト㈱
Author 牛田 隆之（うしだ たかゆき） フューチャーアーキテクト㈱
Twitter @vuls_ja

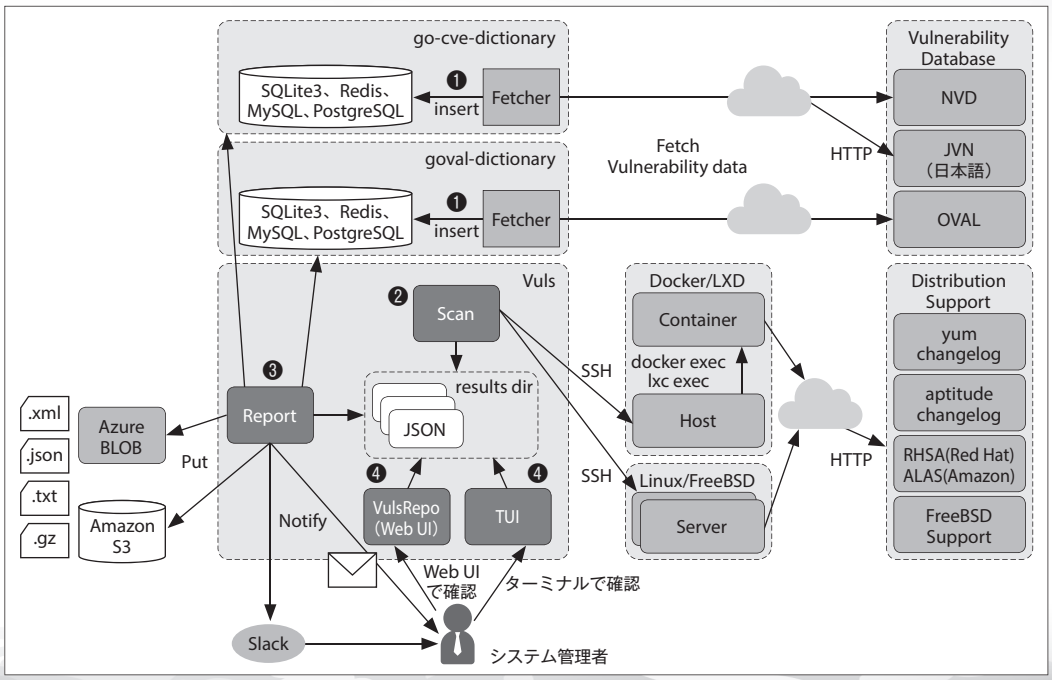
第1章でみなさんのVulsに対する興味がわいたところで、第2章では実際にVulsの実行環境の構築、スキャンの実施、スキャン結果の閲覧までをチュートリアル形式で説明します。ローカル環境のスキャンの手順から説明しますので、お手元のPCですぐにでも試せます。

Vuls Scanの流れ

図1はVulsを用いた脆弱性スキャンの全体像です。図中の❶～❹では次のことが行われます。

❶NVD、JVN、OVALといったインターネットに公開されている脆弱性情報を取得し、ローカルデータベースを構築
❷スキャン対象サーバに接続してコマンドを実行し、ソフトウェア情報や脆弱性情報を取得
❸取得した情報と、❶で構築した脆弱性情報のローカルデータベースを照合し、Slackやメールで通知
❹TUIやVulsRepoなどの可視化ツールで確認

▼図1　Vulsの全体像とスキャンフロー





実行環境構築

今回はUbuntu 16.04上にVulsをセットアップします。環境構築の前に、Vulsの実行ユーザとログ格納用ディレクトリを作成します。

```
Vuls実行ユーザの作成
$ sudo adduser vuls
$ sudo gpasswd -a vuls sudo
$ su - vuls

ログ格納用ディレクトリ作成
$ sudo mkdir -p /var/log/vuls
$ sudo chown vuls /var/log/vuls
$ sudo chmod 700 /var/log/vuls
```

そして、ビルドに必要なGo言語とGit、make、gccをインストールします(**図2**)。Go言語のインストール後、go versionでgoのバージョン(執筆時点でversion 1.8.3)が表示されることを確認してください。

脆弱性情報の取得

NIST(アメリカ国立標準技術研究所)が管理するNVD、IPA(情報処理推進機構)が管理するJVNから[注1]脆弱性情報を取得してローカルデータベースを構築するために、go-cve-dictionaryをインストールします(**図3**)。

次に、各Linuxディストリビュータから公開されている脆弱性情報であるOVALのローカルデータベースを構築するために、goval-dictionaryをインストールします(**図4**)。

続いて、脆弱性情報のローカルデータベースを構築しましょう。

```
$ cd $HOME
$ go-cve-dictionary fetchnvd -last2y
$ go-cve-dictionary fetchjvn -last2y
$ goval-dictionary fetch-ubuntu 16
```

本チュートリアルではNVD、JVNの直近2年

注1) NVD、JVNとも脆弱性情報を管理するデータベース。NVDは英語で、JVNは日本語で情報が管理されている。

▼図2 Go言語、Git、make、gccのインストール

```
Go言語のインストール
$ wget https://storage.googleapis.com/golang/go1.8.3.linux-amd64.tar.gz
$ sudo tar -C /usr/local -xzf go1.8.3.linux-amd64.tar.gz
$ sudo vi /etc/profile.d/goenv.sh   viが起動したら以下の3行を追記して保存
export GOROOT=/usr/local/go
export GOPATH=$HOME/go
export PATH=$PATH:$GOROOT/bin:$GOPATH/bin
$ source /etc/profile.d/goenv.sh
$ go version

Git、make、gccのインストール
$ sudo apt install git make gcc
```

▼図3 go-cve-dictionaryのインストール

```
$ mkdir -p $GOPATH/src/github.com/kotakanbe
$ cd $GOPATH/src/github.com/kotakanbe
$ git clone https://github.com/kotakanbe/go-cve-dictionary.git
$ cd go-cve-dictionary
$ make install
```

▼図4 goval-dictionaryのインストール

```
$ cd $GOPATH/src/github.com/kotakanbe
$ git clone https://github.com/kotakanbe/goval-dictionary.git
$ cd goval-dictionary
$ make install
```

分のデータベースだけを取得しています。全期間の脆弱性データベースを取得する場合はgo-cve-dictionaryのREADME[注2]を参照してください。

また、今回はUbuntu 16のOVALのみを取得しましたが、ほかのOSをスキャンする場合は、スキャン対象サーバのOSとバージョン用のOVALを事前に取得する必要があります。詳細はgoval-dictionaryのREADME[注3]を参照してください。

Vulsのインストールと設定

それでは実際にスキャンしていきます。いよいよVulsの出番です。まずはインストールしましょう(**図5**)。

次に、設定ファイルをTOMLフォーマットで**図6**のように作成します。

最後に設定ファイルが正しく記述できているかのチェックを行います。

```
$ vuls configtest
```

エラーが表示されなければ、スキャンの準備は完了です。

注2) URL https://github.com/kotakanbe/go-cve-dictionary
注3) URL https://github.com/kotakanbe/goval-dictionary



それでは、実際にスキャンしてみましょう。Vulsにはローカルスキャンとリモートスキャンの2種類のモードがあります。

ローカルスキャンは、スキャン対象サーバ上にVulsのバイナリを配置して自分をスキャンするモードです。

リモートスキャンは中央のVulsサーバからほかのサーバに対してSSHで接続してスキャンするモードであり、1台のVulsサーバから複数のサーバをスキャンすることができます。環境に応じて使い分けてください。

ローカルスキャン

まずローカルスキャンを行ってみましょう。

```
$ vuls scan
```

すると、**図7**のような結果が表示されます。

スキャン結果の閲覧

それでは結果を見てみましょう。**図8**のように実行すると、サーバに潜在する脆弱性の総数

▼図5　Vulsのインストール

```
$ mkdir -p $GOPATH/src/github.com/future-architect
$ cd $GOPATH/src/github.com/future-architect
$ git clone https://github.com/future-architect/vuls.git
$ cd vuls
$ make install
```

▼図6　設定ファイルの作成

```
$ cd $HOME
$ vi config.toml
 viが起動したら以下の内容を記述して保存
[servers]
[servers.localhost]
host        = "localhost"
port        = "local"
```

▼図7　スキャン実行結果

```
One Line Summary
================
localhost       ubuntu16.04     18 updatable packages
```

▼図8　Vulsスキャン結果表示

```
$ vuls report -format-short-text --lang=ja
localhost (ubuntu16.04)
=======================
Total: 6 (High:1 Medium:4 Low:1 ?:0)    18 updatable packages
  └(A)          └(B)                        └(C)
CVE-2016-5636           10.0 (High)     CPythonのzipimport.cのget_data関数における整数オーバーフローの脆弱性
(..以下略..)              └(D)                                  └(E)
```

(A)、脆弱性重大度レベルごとの脆弱性数(B)が表示され、アップデート可能なパッケージの数(C)も表示されます。その下の「CVE-2016-5636」が個々の具体的な脆弱性の情報です。10段階で示された危険性(D)と、脆弱性の詳細(E)が書かれています。誌面では省略しますが、「以下略」以降に、6個の脆弱性情報が出力されます。ここから脆弱性情報を確認できますが、次節「スキャン結果の可視化」では、より見やすくする方法も紹介します。

リモートスキャン

ローカルスキャンはVulsを入れた1台のみのスキャンでしたが、リモートスキャンでは一度に複数台のサーバをスキャンすることができます。それでは、リモートスキャンの設定を見ていきましょう。

まずはリモートスキャンしたいサーバへ公開鍵認証でSSHアクセスできるよう設定します。**図9**のようにしてssh-keygenでSSH鍵を作成し、「~/.ssh/id_rsa.pub」の内容をクリップボードにコピーします。

次にリモートサーバにアクセスし、公開鍵を設置します。

```
$ mkdir ~/.ssh
$ chmod 700 ~/.ssh
$ touch ~/.ssh/authorized_keys
$ chmod 600 ~/.ssh/authorized_keys
$ vi ~/.ssh/authorized_keys
```

上記手順の最後でviが起動したら、「~/.ssh/authorized_keys」へ、先ほどクリップボードにコピーした「~/.ssh/id_rsa.pub」の内容を貼り付けて保存します。これで設置完了です。

スキャン実行時にVulsインストール済みサーバの「known_hosts」にリモートサーバのホストキーが登録されている必要があるため、接続確認もかねて一度手動でSSH接続をしておきます(**図10**)。

なお、プロキシ環境下ではリモートサーバ上のsudoersに設定が必要な場合があります。詳しくはREADME[注4]を参照してください。

続いて、Vulsインストール済みサーバにて、

注4) URL https://github.com/future-architect/vuls/blob/master/README.ja.md

▼図9　Vulsインストール済みサーバでの鍵作成

```
$ ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/home/vuls/.ssh/id_rsa):  ←リターン
Created directory '/home/vuls/.ssh'.
Enter passphrase (empty for no passphrase):  ←空のままリターン
Enter same passphrase again:  ←空のままリターン

$ cat ~/.ssh/id_rsa.pub  ←左記コマンドで表示された内容をクリップボードにコピー
```

▼図10　Vulsインストール済みサーバからリモートサーバへアクセス

```
192.168.0.2に対してuser01で接続
$ ssh -l user01 -i /home/user01/.ssh/id_rsa 192.168.0.2
```

▼図11　リモートスキャン用の設定ファイル作成

```
$ vi config.toml  viが起動したら以下の内容を記述して保存
[servers]
[servers.remote01]
host        = "192.168.0.2"
port        = "22"
user        = "user01"
keyPath     = "/home/vuls/.ssh/id_rsa"
```

図6で作成した設定ファイルを図11のように書き換えます。hostはリモートアクセス先のホスト名やアドレスを、portはSSHのポートを記述してください。userにはSSHで接続するときのユーザ名を、keyPathにはサーバへアクセスするための秘密鍵を指定してください。

設定ファイルを編集したあと、正常に記述できているかの確認コマンドを実行します。

```
$ vuls configtest remote01
```

ここまでくれば、あとはスキャンを実行するだけです。今回はリモート先の脆弱性情報が結果として表示されます。

```
$ vuls scan remote01
```

コンテナスキャン

Vulsはホスト OSだけではなく、DockerやLXDで立ち上げた稼働中のコンテナに対してスキャンが可能です(図12)。コンテナスキャンを行うには、設定ファイルを図13のように書き換えます。このように記述すると、192.168.0.2内で起動中のすべてのDockerコンテナをスキャンし、脆弱性情報を表示します。

また、特定のコンテナだけをスキャン、特定のコンテナ以外をスキャンすることもできます。その場合はリスト1、2のように記述してください。コンテナはコンテナID／コンテナ名ともに指定できます。



ここまでの手順でサーバに含まれる脆弱性のスキャンを行い、サマリーレポートを表示できました。Vulsでは、これらのほかにもスキャンした結果の詳細情報を表示するための手段が用意されています。

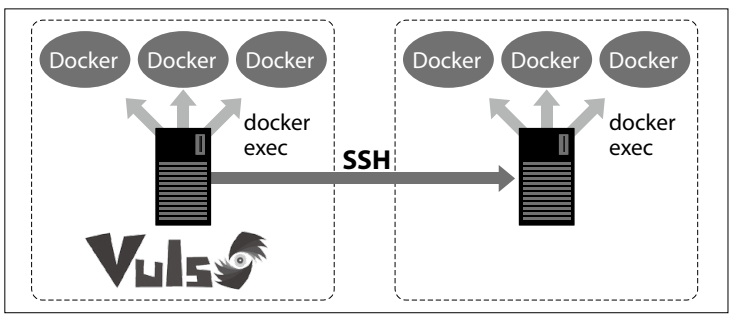
TUIによる表示

Vulsではスキャン結果の詳細を参照するTUIが標準で付属しています。TUI(Terminal-Based User Interface)モードはコンソール上でグラフィカルな画面を表示し、キーボードの方向キーとタブキーのみで軽快にスキャン結果を参照できます(図14)。TUIモードを起動するには、次のコマンドを実行します。

```
$ cd $HOME
$ vuls tui
```

TUIモードから抜けるときは Ctrl + C です。

▼図12　Dockerスキャン時の接続フロー



▼図13　コンテナスキャン用の設定ファイル作成

```
$ vi config.toml   viが起動したら以下の内容を記述して保存
[servers]
[servers.remote01]
host         = "192.168.0.2"
port         = "22"
user         = "user01"
keyPath      = "/home/vuls/.ssh/id_rsa"

[servers.remote.containers]
includes = ["${running}"]
```

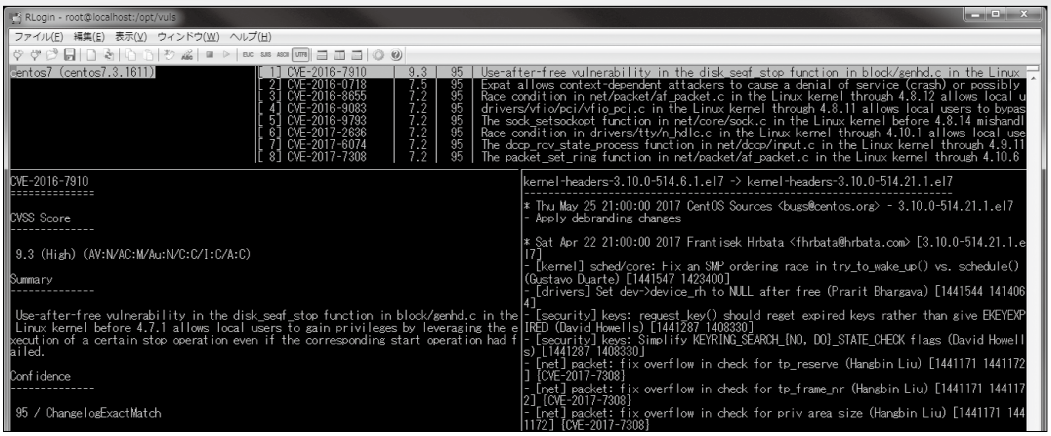
▼リスト1　特定のコンテナだけをスキャン

```
[servers.remote.containers]
includes = ["container01"]
```

▼リスト2　特定のコンテナ以外をスキャン

```
[servers.remote.containers]
includes = ["${running}"]
excludes = ["3ayv35s8g4as"]
```

▼図14　VulsのTUIモードによる脆弱性の表示



VulsRepoによるドリルダウン

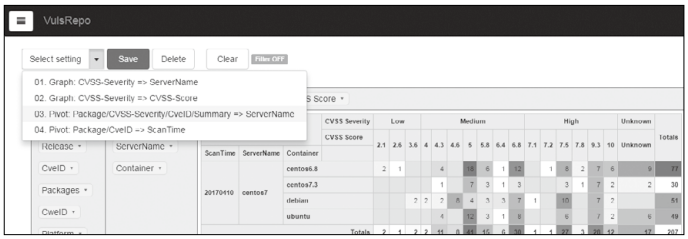
周辺ツールとしてWebブラウザからインタラクティブな操作が行える「VulsRepo」というWeb UIも用意されています。VulsRepoを使うとVulsが検知した脆弱性をExcelに付属しているピボットテーブルのようにさまざまな評価軸で集計／ソートし、自分の見たい情報へとドリルダウンすることができます。

たとえば、Vulsで出力したJSONファイルを読み込むことでサーバ、およびスコアごとに集計したヒートマップを表示できます(**図15**)。

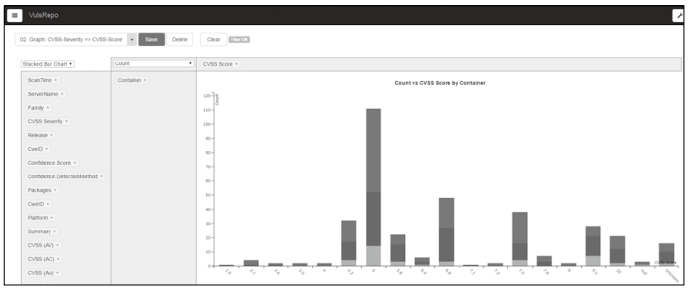
そのほか集計結果をグラフ化(**図16**)したり、脆弱性の詳細情報を表示(**図17**)したりできます。

また、ピボットテーブルの項目をドラッグ＆ドロップすることで集計軸を自由に変更できます。左上のプルダウンには、よく利用されるピボットテーブルの表示設定がテンプレートとして保存されています。また、自

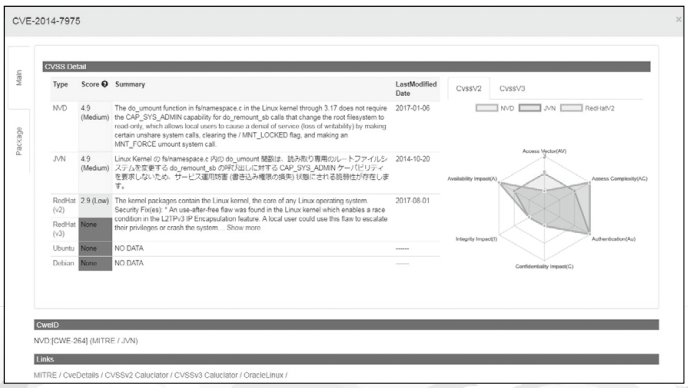
▼図15　VulsRepoのヒートマップ表示画面



▼図16　VulsRepoのグラフ表示画面



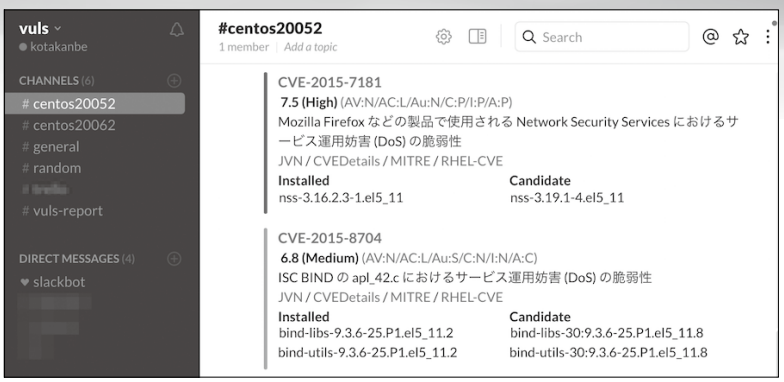
▼図17　VulsRepoの脆弱性の詳細表示画面



分で変更した内容を保存することもできます。脆弱性情報は日本語で表示することも可能です。

VulsRepoはGitHub上にてOSSとして公開しており[注5]、オンラインで試せるデモもあります。Vulsと併せて使用することで検知結果をより柔軟に分析できますので、ぜひ試してみてください。

▼図18　Slackに通知した場合の表示例





メール、Slackへの通知

実際の運用ではcronなどに登録して自動的にスキャンを実行することが多いかと思います。そのため、Vulsではスキャン結果をEメール、またはSlackへ自動通知する機能を持っています。通知を行うためには、まず設定ファイル(config.toml)に接続設定を記載しておく必要があります(**リスト3**、**4**)。

設定を行ったら、次のとおり各オプションを付けて実行します。Slackなら**図18**のように通知されます。

Eメール送信の場合
```
$ vuls report -to-email -lang=ja
```

Slackへ連携する場合
```
$ vuls report -to-slack -lang=ja
```

Eメール、Slackともに脆弱性のサマリーは-lang=jaオプションを付けることで、日本語で表示することが可能です。

差分表示

Vulsの通知機能は、デフォルトでは検知した脆弱性を毎回全件通知します。しかし、システム運用の中ではすぐに対応できない場合もあり、全件通知されると情報が埋もれてしまう場合があります。

そのような場合、-diffオプションを追加することで前回のスキャン結果と比較して差分の

注5) URL https://github.com/usiusi360/vulsrepo

▼リスト3　メール通知する場合の設定内容

```
[email]
smtpAddr  = "smtp.gmail.com"
smtpPort  = "587"
user      = "username"
password  = "password"
from      = "from@address.com"
to        = ["to@address.com"]
cc        = ["cc@address.com"]
subjectPrefix    = "[vuls]"
```

▼リスト4　Slackへ連携する場合の設定内容

```
[slack]
hookURL     = "https://hooks.slack.com/services/abc123/defghijklmnopqrstuvwxyz"
channel     = "#channel-name"
#channel    = "${servername}"
iconEmoji   = ":ghost:"
authUser    = "username"
notifyUsers    = ["@username"]
```

情報だけを通知することができます。

```
$ vuls report -to-email -lang=ja -diff
```



CPE名を用いた検知

VulsはOSのパッケージ構成情報を収集する以外に、CPE(Common Platform Enumeration)[注6]と呼ばれるハードウェア、OS、アプリケーションなどのプラットフォームを識別するための名称を使用することでNVD、JVNのデータとマッチングすることができます。

これによりyumやaptなどパッケージ管理システムを使わずに、ソースからビルドしたものやプログラミング言語のライブラリ、ネットワーク機器の脆弱性をチェックできます。

CPEを利用するには設定ファイル(config.toml)で**リスト5**のように設定します。プロダクトごとのCPEは、NVDの「Search Common Platform Enumerations(CPE)」のサイト[注7]から検索できます。なお、Vulsで使用できるCPE Naming Formatは「2.2」になります。

OWASP Dependency Checkとの連携

利用しているライブラリが多い場合、Vulsで開発言語のライブラリのCPEを1つずつ指定するのは現実的ではないかもしれません。そのような場合は、OWASP Dependency Checkと組み合わせて使用できます。OWASP Dependency Checkはプログラミング言語のライブラリを特定(CPEを推測)し、公開済みの脆弱性を検知するツールです。

Vulsと組み合わせて使用することで、Vulsの持つSlack、Eメールへの通知機能が使えたり、OWASP Dependency Check単体では対応していない日本語でのレポート出力が可能になったりします。VulsのREADME[注8]に設定方法が掲載されていますので、興味のある方は参考にしてください。



今回はチュートリアルということで、ソースをダウンロードしてVulsをビルドする手順を紹介しましたが、Dockerイメージやクラウドベンダが提供するテンプレートやスクリプトを使用して導入することもできます。

- Vuls Dockerイメージ[注9]
- IDCF Vulsコミュニティテンプレート[注10]
- さくらインターネット Vulsスタートアップスクリプト[注11]

導入環境に合わせて、これらも活用していただければと思います。 SD

注6) URL https://www.ipa.go.jp/security/vuln/CPE.html
注7) URL https://nvd.nist.gov/products/cpe/search
注8) URL https://github.com/future-architect/vuls/blob/master/README.md#usage-integrate-with-owasp-dependency-check-to-automatic-update-when-the-libraries-are-updated-experimental
注9) URL https://github.com/future-architect/vuls/tree/master/setup/docker
注10) URL http://blog.idcf.jp/entry/idcf-vuls2
注11) URL https://cloud-news.sakura.ad.jp/2017/06/27/startupscript-vuls/

▼リスト5　CPEを使う場合の設定例(Ruby on Railsを対象とした場合)

```
[servers]
[servers.172-31-4-82]
host         = "172.31.4.82"
user         = "ec2-user"
keyPath      = "/home/username/.ssh/id_rsa"
cpeNames = [
  "cpe:/a:rubyonrails:ruby_on_rails:4.2.1",
]
```

第3章 Vulsとさまざまなプロダクトとの外部連携



Author 牛田 隆之（うしだ たかゆき）フューチャーアーキテクト㈱
Twitter @vuls_ja

Vulsはちょっとしたカスタマイズを加えることによりさまざまなプロダクトと連携でき、Vuls単体で使用するよりもずっと楽に運用できます。本章では監視システムとの連携、Excelレポート作成、BI系ツールによるダッシュボード作成、クラウドサービスとのAPI連携について事例を交えながら紹介します。

監視システム（Zabbix）との連携

Vulsは自身の設定ファイル「config.toml」に、スキャン対象とするホスト情報（ホスト名、IPアドレス）を記述することでスキャンができます。しかし、すでに監視システムが導入されている場合、Vulsと監視システムで二重にホスト情報を持つことになり、管理が猥雑（わいざつ）になります。そこで今回は、OSSの統合監視ソフトウェア「Zabbix」[注1]を例として、Zabbixに登録されているホスト情報をAPI経由で抜き出し、Vulsの設定ファイルを自動生成することでホスト情報の管理を一元化します。

また、第2章にてVulsで脆弱性を検知した場合にEメール、Slackへ通知する方法を紹介しましたが、すでに監視システムを中心に運用フローが組まれている場合には、既存の監視システムからアラートを発報することで、通知フローを一元化するほうが望ましいことがあります。そのため、Vulsで検知した結果を集計してZabbixに連携し、検知数に変化があった場合のみアラート発報するようにします。

本節で紹介する内容を実施することで次のフローを自動化できます。

①Zabbixからホスト情報を抜き出し、Vulsのスキャン設定ファイルを自動生成
②Vulsでスキャン実行（JSON形式のスキャン結果ファイルを出力する）
③JSON形式のスキャン結果ファイルを集計し、Zabbixに送信

今回は、実際に動作させるためのスクリプトを紹介しながら説明していきます。

Zabbixのホスト情報からVulsの設定ファイルを自動生成

まずは、スキャン対象ホストで共通の設定部分を「config.toml.master」として作成します（**図1**）。このファイルの内容は後述するスクリプトの中でZabbixから抜き出した情報と結合されます。

次にZabbixからホスト情報を抜き出すスクリプト「create_config.sh」をダウンロードします（**図2**）。

このスクリプトは、Zabbix上に登録されてい

▼図1　config.toml.masterの作成

```
$ cd /opt/vuls
$ vi config.toml.master
 viが起動したら以下の内容を記述して保存
[default]
port        = "22"
user        = "username"
keyPath     = "/home/username/.ssh/id_rsa"
 通知が必要な場合は、[mail]または[slack] セクションもここに記述
```

注1）URL https://www.zabbix.com/jp/

るホストのうち、Zabbix Agentのステータスが「正常」、かつホストのステータスが「有効」になっているものをAPIで抽出します。さらにホストの「エージェントのインターフェース」の中から1つめのIPアドレスだけを抜き出し、Vulsの設定ファイル形式に整形します。

なお、スクリプトの**リスト1**の部分には、Zabbixへ接続するための情報が記載されているので、環境に合わせて変更してください。

Zabbixに登録されているホストの中に、Vulsでスキャン対象としたくないものがある場合は、次のように「ignore.list」を作成し、スキャン対象外とするホスト(ホスト名かIPアドレス。部分一致も可)を記載します。

```
ignore.list設定例
$ vi ignore.list
viが起動したら以下の内容を記載して保存
sw001
db
192.168.10
```

最後にスクリプトを実行すると、設定ファイルが作成されます(**図3**)。

ここまでの手順でZabbixのAPIから自動的にホスト情報を抜き出し、Vulsの設定ファイルを自動生成することができました。Zabbixのホストディスカバリ機能を使って自動でホスト追加を行っている場合、今回のスクリプトをcronなどで自動実行しておくことでVuls側でも自動的にスキャン対象ホストを追加することができます。

Vulsスキャン～Zabbixへアラート通知

ここからはVulsでスキャンを行い、Zabbixへ連携して通知を行います。すでに、シェルスクリプトとして公開していますので、それをダウンロードしてください(**図4**)。

なお、スクリプト内部でJSON形式のファイルをパースする「jq」コマンドとZabbixへデータを送信する「zabbix_sender」コマンドを使用しています。インストールされていない場合は、別

▼図3　create_config.shの実行

```
$ ./create_config.sh  ←実行

$ cat config.toml  ←作成された設定ファイルを参照
[default]
port        = "22"
user        = "username"
keyPath     = "/home/username/.ssh/id_rsa"

↓↓↓ 自動生成されている ↓↓↓
[servers]
[servers.web001]
host        = "192.168.0.1"

[servers.app001]
host        = "192.168.0.2"
```

▼図2　create_config.shのダウンロードと実行権限の付与

```
$ wget https://github.com/usiusi360/vuls_autoscan_for_zabbix/raw/master/create_config.sh
$ chmod 755 create_config.sh
```

▼リスト1　create_config.shのZabbix接続設定部分

```
#!/bin/bash
ZABBIX_SERVER="zabbix-server"  ←環境に合わせて変更
ZABBIX_USER="Admin"  ←環境に合わせて変更
ZABBIX_PASS="hogehoge"  ←環境に合わせて変更

VULS_HOME=`cd $(dirname $0) && pwd`
(..以下略..)
```

▼図4　vuls_autoscan_for_zabbix.shのダウンロードと実行権限の付与

```
$ cd /opt/vuls
$ wget  https://github.com/usiusi360/vuls_autoscan_for_zabbix/raw/master/vuls_autoscan_for_zabbix.sh
$ chmod 755 vuls_autoscan_for_zabbix.sh
```

途インストールしておいてください。また、スクリプトの**リスト2**の部分にてZabbixサーバのアドレスを指定していますので、環境に合わせて変更が必要です。

スクリプト内部では次のことを実施します。

①脆弱性情報(NVD/JVN)の最新化(go-cve-dictionary)
②Vulsでスキャン(vuls scan)
③Zabbixへ結果連携(zabbix_sender)

③では、Vulsが出力したスキャン結果ファイルからホストごとの脆弱性の個数と深刻度の最大値を、zabbix_senderコマンドの引数に入れて送信します。

Zabbix側の設定

スクリプトの配置ができたら、次にZabbix側の設定を行います。GitHub[注2]に、Zabbixのデータ収集設定(アイテム)、およびアラート設定(トリガー)を記述したテンプレート「Template_Vuls.xml」が用意してあります。こちらをダウンロードし、Zabbixにインポートしたあと Vulsのスキャン対象ホストへテンプレートを適用します。

cronへ登録

最後に**リスト3**のようにcronへ登録し、シェルスクリプトを自動実行します。

なお、cronから実行すると「/etc/profile.d/」以下の環境変数は読み込まれないため、goの環境変数を読み込むにはbash -lを付ける必要があります。

正しく設定されていれば、Vulsのスキャン結果をもとにZabbix側で「検知したCVEの数」と「検知したスコアの最大値」を収集できます(**図5**、**6**)。そしてZabbixのトリガー設定により、前回チェック時から検知数が「増加した場合」にアラートとして検知できます。

今回はZabbixを例として連携事例を紹介しましたが、ほかの監視ソフトウェアでも同様に簡単なスクリプトで仲介することでVulsと連携できます。そして、脆弱性チェックをシステム監視と一緒に統合運用できます。

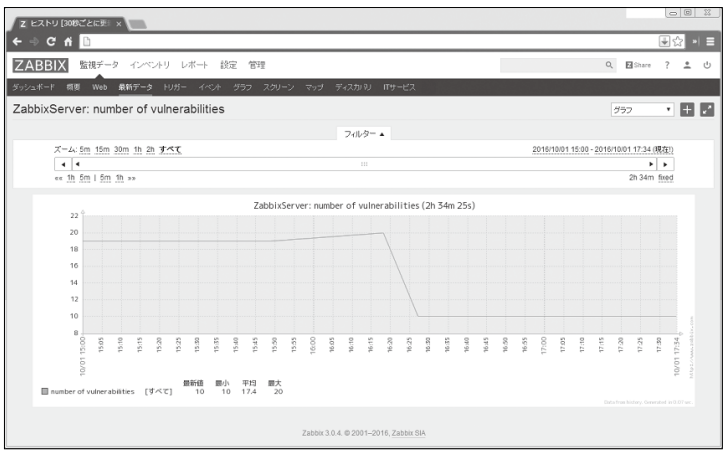
▼リスト2　vuls_autoscan_for_zabbix.shのZabbix接続設定部分

```
#!/bin/bash
ZABBIX_SERVER="localhost"   ←環境に合わせて変更
(..以下略..)
```

▼図5　Zabbixでの収集結果



▼図6　Zabbixでの収集結果(グラフ)



▼リスト3　cronの設定(/etc/crontab)

```
0 13 * * * vuls-user bash -l /opt/vuls/vuls_autoscan_for_zabbix.sh > /tmp/vuls.log 2>&1
毎日13時00分に、vuls-userユーザにてvuls_autoscan_for_zabbix.shを実行する
```

注2) URL https://github.com/usiusi360/vuls_autoscan_for_zabbix

Excelで静的レポートを作成

システム運用の一環として、脆弱性のチェック状況を定期的に報告する場面は多いかと思います。そのような場合、データの閲覧者は運用担当以外、たとえば社外の顧客や自部門外の組織となる場合があり、レポート形式での提出が必要になります。

VulsではJSON形式でデータを出力できますので、本節ではそのデータをCSV形式に変換するログコンバータ「vuls-log-converter」[注3]を使用して、データの受け渡しがしやすく、導入が容易なExcelでレポートを作成します。

vuls-log-converterのインストール

事前にNode.js(JavaScriptの実行環境)とnpm(Nodo.jsのパッケージ管理ツール)をインストールしておいてください[注4]。なお、Node.jsはv4以上が必要です。

npmで、Vulsのログコンバータ「vuls-log-converter」をインストールします。

```
$ npm i -g vuls-log-converter
$ vulslogconv -h ←使い方を表示
```

変換実行

vulslogconvコマンドでCSVに変換します(**図7**)。-iオプションでVulsが出力したJSONファイルが入っているディレクトリを指定し、-oオプションで出力先のファイルを指定します。入力元にディレクトリを指定すると中に入っているJSONファイルをすべて変換し、1つのCSVファイルに出力します。

ExcelでCSV取り込み→レポート作成

変換が完了したCSVファイルをExcelに取り込んでレポートを作成します(**図8**)。ピボットテーブルやグラフを使ってわかりやすいように整形します。データ取り込み用のシートを分けておくと、データを入れ替えてピボットテーブルやグラフの更新を行うだけで、グラフなどを作り変えられるため毎月の作成が楽になります。

Kibana＋Elasticsearchでダッシュボード作成

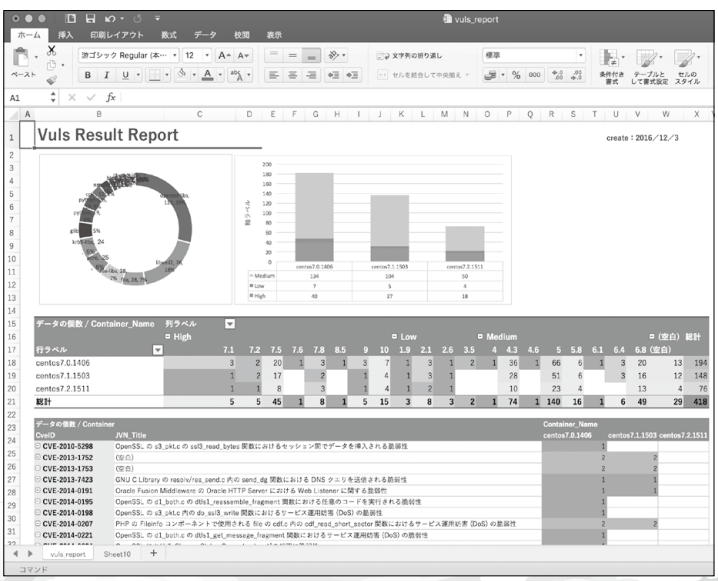
システムの規模が大きいとVulsが出力するログの量も膨大になり、Excelで処理することが難しくなります。また、Vuls以外で管理している構成情報と組み合わせたり、ドリルダウンしながらインタラクティブに分析したりしたい場合、BI系のデータ可視化ツールを使うと分析の幅が広がります。

今回は、ElasticsearchとKibanaを使って

▼図7　vulslogconvによる変換実行

```
$ vulslogconv -t csv -i /opt/vuls/results/current/ -o ./output.csv
```

▼図8　Excelでのレポート作成例

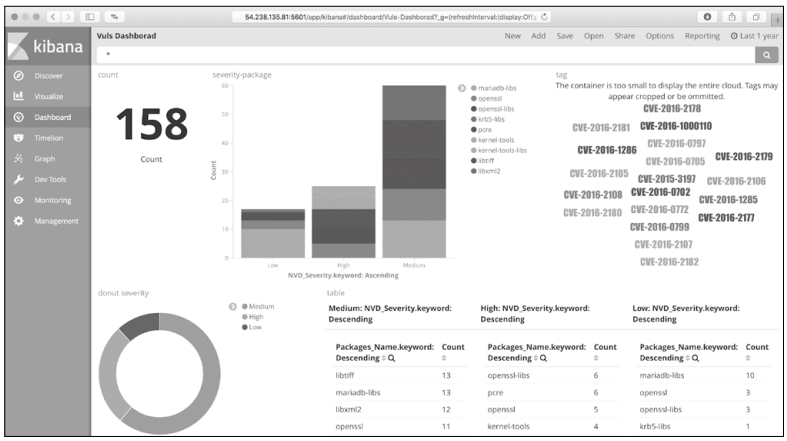


注3）URL https://github.com/usiusi360/vuls-log-converter
注4）URL https://nodejs.org/ja/download/package-manager/

▼図9　Elasticsearchへのデータのインポート

```
$ vulslogconv  -t els -i /opt/vuls/results/current/ -e https://search-vuls-hogehoge.ap-northeast-1.es.amazonaws.com/
```

▼図10　Kibanaによるダッシュボード例



Vulsのログを可視化したいと思います。なお、Kibana + Elasticsearchの操作はほかに資料がたくさんあるため要点のみ記載します。

Elasticsearchへインポート

前節で使用した「vuls-log-converter」を使います。Elasticsearchは、今回の場合、「Amazon Elasticsearch Service」で作成した環境を使います。vulslogconvコマンドの引数にElasticsearchのエンドポイントを指定することで、VulsのJSONファイルを変換しつつインポートを行います(**図9**)。

Kibanaで可視化

Elasticsearchにデータをインポートできたら、Kibanaでインデックスを指定します。

・index name or pattern：vuls_index
・Time-field name：ScannedAT

データを取り込めていることを確認できたら、Kibana上で普段確認したい情報をダッシュボードとして作成します(**図10**)。

今回は「Kibana + Elasticsearch」で作成しましたが、BI系のツールであればほとんどの場合、CSV形式のインポートは対応しているかと思います。自分にあったツールに取り込んでダッシュボードを作成し、普段から分析することですばやい脆弱性対応につながるのではないかと思います。

AWSとの連携

config自動生成

Zabbixとの連携のところでホスト情報の一元化について紹介しましたが、AWS環境の場合も同様にマネージメントコンソール上のEC2インスタンス一覧をAPIで出力し、Vulsのコンフィグファイルを作成するツールが公開されています[注5]。

また、ほかのクラウドサービスでも、インスタンス情報の一覧を抜き出すことは簡単にできるため簡易なスクリプトでVulsへ連携することが可能です。SD

注5) URL https://github.com/ohsawa0515/ec2-vuls-config

第4章 Vuls誕生秘話 ——OSSをバズらせるには



Author 神戸 康多（かんべ こうた） フューチャーアーキテクト㈱
Twitter @vuls_ja

本章ではVulsの生みの親が、着想から、実際の開発、公開まで、Vulsの生い立ちを紹介します。また公開後、OSSであるVulsの認知度をどのように上げてきたのかも併せて語ります。



苦悩と憤り、そして訪れたVulsが生まれる転機

脆弱性運用への憤り

第1章で説明したように、手動での脆弱性管理は非常にたいへんな作業です。それに、対策漏れのリスクを考えると、本来は人手でやるべきではないと思いませんか？　筆者は自分の経験から、心の底から自動化すべきと思いました。

数年前、100台以上のサーバ環境の運用をしていたときの話です。運用していた環境はさまざまな種類、バージョンのOS、プログラミング言語、ミドルウェアから構成され、複雑で混沌としていました。しかも会社の重要情報を扱うものであり、高いセキュリティレベルが求められました。脆弱性管理、アップデートの運用を真面目にやろうと考えた筆者でしたが、手動での脆弱性管理を毎日続けるのは不可能であると3日で理解しました。膨大な脆弱性情報と向き合い、自分が管理するシステムに必要な情報を探し出し、パッチを適用する。果てしないうえに生産性のない作業の繰り返しに、絶望的な気持ちになりました。そして、その絶望はすぐに憤りに変わりました。

新しい脆弱性が公開されたタイミングで、「あなたの管理しているこのサーバの、このソフトウェアに、こういう脆弱性が発見されて、その危険度はこれくらいですよ」と、システムが教えてくれてもいいのに、と。こうした思いがのちにVulsの開発の動機につながっていったのです。

転機

ちょうどそのころ筆者は、入社10年目でもらえるリフレッシュ休暇を利用し、バックパックを背負いインドとネパールに1ヵ月ほどの一人旅に出ました。ネパール西部でのヒマラヤ・トレッキングのあと、南部のルンビニというブッダ生誕の地を訪れた筆者は、そこであるお坊さんと出会います。これが、OSS開発に目覚めることになったきっかけでした。

ルンビニには各国の寺があり、旅行者が寺に宿泊して修行できます。せっかくの機会ですから、その中で一番厳しいと言われる通称「日本寺」に行ってみることにしました。寺には80歳の日本人のお坊さんがいて、普段はアフリカの寺で布教していること、世界平和が夢であり、心の底から実現しようとしていること、筆者も体験した修行、質素な食事をこれまで60年以上毎日のように続けてきて、世界平和を実現するために今後も続けていくつもりだということを語ってくれました。

そのお坊さんと話をする中で、その人の信念の強さ、本気さにショックを受けました。今ま

での人生を振り返り、基本的に自分のことしか考えてこなかった、自分の未熟さ、視野の狭さを痛感しました。年齢も生活している場所も、日々何をしているかもまったく異なりますが、お坊さんとの出会いに筆者は大きな影響を受けました。

インドに移動してからもそのことが頭から離れず、「自分が心の底から世の中のために実現したいと思うことは何なんだろう？」「どのような方法で世の中に貢献できるのだろう？」とガンジス川のほとりで数日間考えました。

その結果、あるアイデアに行き着きました。エンジニアである筆者にとって、一番効果的な貢献方法は、筆者の技術を使って人の役に立つツールを作り、それを普及させ、使った人を幸せにすること。そのツールで幸せになった人びとの総和が世の中への貢献度である。つまり良いツールを作成してOSSとしてバズらせれば、世の中に貢献したと言える、という考えでした。この考えと過去に脆弱性対策で感じたいろいろな思いが組み合わさり、脆弱性スキャナの開発、公開を決意したのです。



捨て身での3ヵ月集中開発

固い決意を胸に帰国した筆者は、さっそく聞き込みをしました。その結果、やはりシステム管理者は脆弱性管理にとても困っていることがわかりました。手応えを感じた筆者は、会社の技術部隊のリーダーに、手動での脆弱性管理のたいへんさと、作ろうとしている脆弱性スキャナのしくみ、有用性と実現可能性を説明し、「開発に専念したいので3ヵ月間ください」と願い出ました。

筆者の熱い思いに対し、会社はVulsの開発に業務として専念することを許可し、要望どおり3ヵ月の時間を与えてくれました。会社が筆者を信じて与えてくれたチャンスです。何が何でも結果を出さなければいけない。エンジニア魂に火がつきました。こうして、他の一切の業務をブロックし、自宅にこもって自分を追い込み、ひたすら作り込む日々が始まりました。

この3ヵ月で良いものができなければ、まったく意味がありません。その場合は責任を取って自らクビになる覚悟でした。あえて退路を断ったのです。筆者はこれまでの人生で一度もないくらいに本気でした。自分の意志で朝から晩まで、平日休日問わず開発を続けました。週3回のサウナが唯一の息抜きと言えるくらい開発に没頭しました。サウナで極限まで汗を絞り出し、水風呂で一気に冷却する。これを3回繰り返せば、どんなに精神的に滅入っていてもまるで生まれ変わったように生き返ります。本音を言えば、作ったものが広く使われるか確信を持てない状況の1人での開発は、精神的に本当に辛かったです。でも、燃え尽きましたと自ら途中で投げ出すわけにはいかない。Vulsの開発は自分自身との闘いでした。

そして公開へ

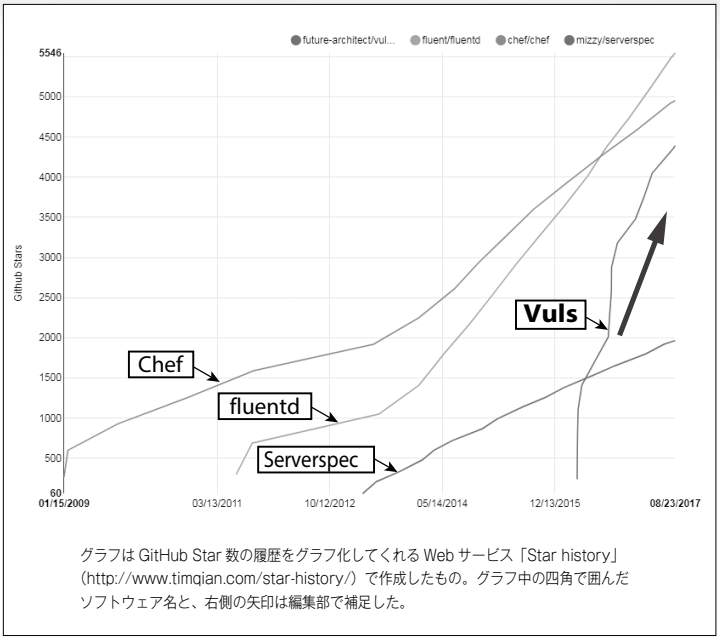
こうして開発は順調に進み、開発スタートから3ヵ月後の2016年4月1日、公開日を迎えました。良いものができたという自信はありました。しかしOSSは使われなければ意味がありません。GitHubには現在3,000万以上のプロジェクトがあると言われていますが、その中でなんとか目立たなければなりません。認知度を上げるために、どんな方法で宣伝すればいいのかを必死に考えました。

日本から攻めるか、海外から攻めるか。日本と英語圏でエンジニアの数を比較すると、英語圏のほうが圧倒的に多く、Twitterのフォロアー数、Facebookグループ、Reddit[注1]の記事数は数十倍以上の規模であることがわかりました。

まずは分母が多い英語圏から攻めることに決

注1）アメリカ発の巨大ソーシャルブックマークニュースサイト。

▼図1　Vulsほか3製品のGitHub Star数の推移



グラフはGitHub Star数の履歴をグラフ化してくれるWebサービス「Star history」（http://www.timqian.com/star-history/）で作成したもの。グラフ中の四角で囲んだソフトウェア名と、右側の矢印は編集部で補足した。

め、READMEを英語で記述しました。READMEの冒頭部分は非常に重要です。多くの新しいライブラリが日々話題になっていますが、見るほうも忙しいので吟味には時間をかけられません。READMEを読んでわずか数十秒で、このツールはこれ以上深く調べる価値があるかどうかを判断します。ですから、とくに冒頭の部分は、図を用いてビジュアルに訴えるなど丁寧に記述しました。また、英語圏で宣伝する以上、GitHub Issueなどの質問も英語で来ますので、なるべく質問が出ないように詳しく説明するよう心がけました。こうして、バグフィックスや機能追加に充てる時間を十分に確保できるようにしました。

さまざまなSNSやWebサイトで宣伝するなかで、もっとも効果があったのは、RedditとTwitterでした。Redditを見たTwitterフォロアー数万人がツイートし、Vulsが一気に世界中へ拡散しました。さらに、リツイートやHacker Newsなどで拡散が進み、GitHub Starがどんどん増えていきました。公開数日後には英語や中国語、フランス語、ドイツ語、スロバキア語などいろいろな言語でVulsを紹介するブログが出現しました。その結果、公開後2週間で1,000ものGitHubスターが集まり、最高のスタートを切れました（**図1**）。

Vulsは公開から約1年半で、約4,300GitHub Star、約600Commit、約150Issue、約300PullRequestという活発なプロジェクトとなっており、着実に進化しています。また、VulsのSlackコミュニティ[注2]も全体で400名以上、日本語のチャンネルで200名以上が参加し、いまや大きなコミュニティとなっています。次章で紹介するように、これからも着実に進化させていきますので、Vulsの今後にもご期待ください。



本章ではVulsの開発秘話、裏話を紹介しました。強い信念と独自のアイデアがあり、どのように実行するか次第で、エンジニアの可能性は無限に広がります。

自作のOSSをバズらせ、世の中に貢献したいと思われている方は、本章で紹介した「世界でバズらせてから日本で広める」やり方をぜひ試してみてください。また、Vuls、VulsRepoを気に入った方はGitHubスターをつけていただければ、それを励みに今後さらに発展させていきます。 SD

注2）「Vuls-Slackチーム参加フォーム」
URL http://goo.gl/forms/xm5KFo35tu

第5章 実際の運用と、今後のVulsの進化



Author 福田 鉄平（ふくだ てっぺい） フューチャーアーキテクト㈱
Twitter @vuls_ja

前章までででセキュリティパッチ（以下、パッチ）管理にまつわる課題と、Vulsの利用方法について紹介しました。本章ではVulsの実際の運用や、今後のVulsの進化について紹介します。



継続的セキュリティ運用のベストプラクティス

Vulsで脆弱性を検知してパッチを適用するという実際の運用においての、2つの重要なしくみ作りについて紹介します。

パッチを適用しやすいしくみ

Vulsは脆弱性を検知しますが、そのあと実際にパッチを適用する際の判断は運用者に任されています。しかし、パッチの適用によりシステムに障害が起きる可能性があるため、検証には多くの時間を費やす必要があります。その結果パッチ適用が遅れ、セキュリティインシデントにつながってしまう恐れがあります。

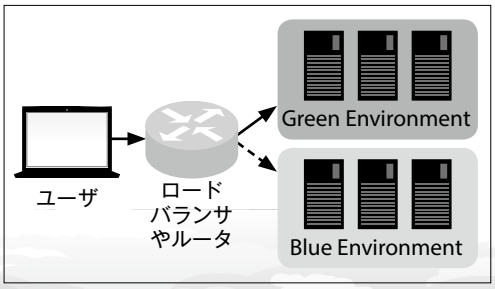
そのため、アップデートを適用しやすいシステムの構築が重要だと考えられます。たとえば、数年前から話題に上がることの多くなった「Immutable Infrastructure」を採用することで、パッチを適用しやすくなります。Immutable Infrastructureでは、サーバを一度構築したら最後、変更を加えません。つまりインフラを“使い捨て”にします。必要に応じてすぐにサーバを用意し、不要になったサーバは破棄します。

このしくみを実現するためにDockerなどが利用されることが多いです。Dockerを使うと、コンテナを誰がどこで起動しても、理論上は同じ環境を構築できます。パッチを適用して検証したうえで、そのコンテナを本番にデプロイすれば、予期せぬ問題は発生しにくくなります。

また、Immutable Infrastructureの利点として「Blue-Green Deployment」を採用できる点も挙げられます。Blue-Green Deploymentでは、新しいアプリケーションコードのためのサーバ群をすべて複製し、ロードバランサやルータなどで切り替えることでデプロイします（**図1**）。こうすることで、万が一問題が発生した場合にも、即座にロールバックすることができます。

パッチを適用しやすいシステムを構築することで、Vulsをより効率的に利用していきましょう。

▼図1 Blue-Green Deployment



パッチ適用対象自体を減らすしくみ

Vulsはインストール済みのソフトウェアの脆弱性を検知しますが、運用環境の中にはインストールされていても利用されていないソフトウェ

アも存在します。実際には利用されていないソフトウェアだとしても、検知された脆弱性の深刻度が高ければ、本当に利用されていないことを運用者は確認する必要があります。この作業は負担も大きく無駄ですので、そもそも本番で利用するサーバにインストールするソフトウェアは最小限に抑えるべきです。

前述したDockerなどを使う場合は、インストールされているソフトウェアが最小限であるAlpine Linux[注1]などのイメージを利用すると良いです。また、開発段階ではビルド用のソフトウェアが必要な場合もありますが、最近リリースされたDockerのマルチステージビルド機能を使うことで、本番用のコンテナにインストールするソフトウェアを減らすことができます。マルチステージビルド機能では、開発用イメージでビルドした生成物を本番用イメージにコピーできるため、本番用のコンテナにビルド用のソフトウェアをインストールする必要がなくなります[注2]。



Vulsは現在でも十分便利なツールではありますが、まだまだ多くの機能追加が予定されています。その中のいくつかをここで紹介します。

パッチ未公開の脆弱性

現在のVulsの実装では、公開されたパッチをもとにサーバ内に存在する脆弱性の検知をしています。深刻度の高い脆弱性のパッチは迅速に適用する必要があるため、パッチの有無と脆弱性の深刻度を把握することは非常に重要です。

しかし脆弱性によっては、影響ありと判明してからパッチが公開されるまで時間がかかるものも存在します。その間は公開された回避策を実施するなどして一時的な対応をする必要がありますが、現在のVulsではこれらの脆弱性は検知されません。たとえばRed Hatでは、パッチが公開されていない脆弱性に対する状況を示す「State」の値が5つあります[注3]。

▼図2　Red HatのAffected Packages Stateの一例[注4]

Affected Packages State

Platform	Package	State
Red Hat JBoss Web Server 3.0	curl	Fix deferred
Red Hat JBoss Core Services 1	curl	Affected
Red Hat Enterprise Linux 7	curl	Will not fix
Red Hat Enterprise Linux 6	curl	Will not fix
Red Hat Enterprise Linux 5	curl	Will not fix
RHEV-M for Servers	mingw-virt-viewer	Affected

- Affected
- Not affected
- Fix deferred
- Under investigation
- Will not fix

これらは、公表された脆弱性がRed Hatのパッケージにどう影響するかを示すものです。**図2**のようにOSのバージョンなどによっても異なります。「Affected」や「Fix deferred」は、影響を受けることは判明しているがパッチがまだ公開されていないことを示します。「Will not fix」は、影響があることは判明しているが修正する予定がないことを意味します。おもに深刻度の低い脆弱性が「Will not fix」になりますが、環境によっては影響が大きい脆弱性も存在します。

これらはOS設定の変更などにより回避・緩和できるものも多いため、検知されればパッチが未公開でも対応できます。そこで、Vulsでは今後、パッチ未公開の脆弱性も検知する機能が実装される予定です。具体的には、Red HatやDebianの各脆弱性の対応状況をSecurity Tracker[注5]などから取得し、影響があるとされ

注1）URL https://hub.docker.com/_/alpine
注2）正確には以前にも、頑張ればイメージを分離できましたが、このマルチステージビルド機能により楽になりました。
注3）URL https://access.redhat.com/blogs/product-security/posts/2066793
注4）URL https://access.redhat.com/security/cve/cve-2016-8615　図2は左記URLの2017年8月現在のもの。
注5）URL https://access.redhat.com/security/security-updates/#/cve
URL https://security-tracker.debian.org/tracker

ている脆弱性はパッチが未公開でも脆弱性として検知します。この機能により、パッチ公開までの期間に一時的な対応をするべき脆弱性や、回避策しか存在しない脆弱性も洗い出されるため、より迅速な対応が可能になります。

攻撃コードの有無

脆弱性の対応をする際、緊急度を決める重要な要素の1つとして、攻撃コードの有無があります。攻撃コードはセキュリティ界隈ではProof of Concept(PoC)と呼ばれることが多いため、以下ではPoCとします。脆弱性自体の深刻度は高くても実際には攻撃につなげるのが難しく、簡単にPoCを作成できない場合は、緊急度を下げて対応することも多いかと思います。

つまり、Vulsでは検知した脆弱性のCVSS基本値[注6]を表示していますが、基本値が高くてもPoCの有無によっては緊急度が低くなり得るということです。そこで、VulsではPoCも脆弱性とともに表示する予定です。具体的にはThe Exploit Database[注7]やMetasploit[注8]を監視して、関連するPoCを検出します。この情報をもとに、セキュリティ担当者に本来の緊急度を判断してもらえればと思います。

そのほか今後の方向性について

誌面の都合上、残りは簡単な紹介に留めます。

まず、現在のVulsはサーバにインストールされているソフトウェアの脆弱性を検知しますが、中にはインストールされていても使われていないものも含まれます。その判断に役立てるため、今後はどのパッケージがどのプロセスで利用されているかを取得し、必ず使われているパッケージを判別できるようにする予定です。

次に、OSのパッケージマネージャで管理されていないパッケージ(JavaやRubyのライブラリなど)は、今のところCPEでの検知がおもになっています。第2章で説明したOWASP Dependency Checkと組み合わせることで、Javaと.NETのライブラリは検知されますが、この2つ以外は実験的なサポートとなっています(2017年7月現在)。今後はほかのツールと組み合わせるなどして、対応可能なライブラリを増やしていく予定です。

ほかにも高速化や検知精度の向上など、多くの機能が追加される予定ですのでご期待ください。もしほかにも要望などあれば、IssueやPull Requestなどをお待ちしています。Slack(第4章参照)では日本語での相談も可能です。



検知後の判断・運用をサポートするSaaS

前節までにでVulsのOSSについて説明してきました。ここでは現在開発中のVulsのSaaS版について簡単に説明します。

検知後の対応

Vulsは脆弱性の情報収集と対象調査のためのツールになっています。つまり、どのサーバにどの脆弱性があるかを検知してくれますが、そのあとの対応についてはサポートしていません。そこで、VulsのSaaSではOSSで検知された結果をもとに、そのあとの対応の負担を減らす機能を提供します。つまり、SaaS版の対応範囲としては**図3**のようになります。具体的には次の機能などが提供されます。

・対応者のアサイン
・対応期限の設定
・対応内容のコメント

これらの機能により、検知後の対応フローが改善されます。

また、CVSSには基本値以外にも現状値や環境値が存在します。これらは**表1**のように、現状評価基準や環境評価基準により算出される値になります。前述したPoCの有無は現状値に影

注6) Common Vulnerability Scoring System：共通脆弱性評価システム。「基本値」は脆弱性そのものの特性を評価する基準(ほかの基準については後述)。
注7) URL https://www.exploit-db.com
注8) URL https://www.metasploit.com

響します。NVDやJVNでは基本値のみ提供されますが、基本値はあくまで脆弱性の深刻度であり、自組織への影響度を測るためには現状値や環境値を計算する必要があります。

SaaS版では、自組織の環境を事前に設定しておくことで、環境に合わせたCVSSの全体的評価値を自動で再計算します。機密性・完全性・可用性の点から、対象システムのセキュリティ要求度なども考慮して評価するため、脆弱性の影響を大きく受けるシステムが数値として見えるようになります。

SaaSの追加機能

脆弱性スキャンに関してもSaaS版にしかない機能がいくつか存在し、その中でも大きいと考えられるのはWindows対応です。SaaS版ではVulsによるWindowsのスキャンも可能になります。Windows管理用のツールとしてWindows Server Update Services(WSUS)などもありますが、WSUSだと深刻度がわかりにくい、検索がしにくい、Linuxサーバと一括で管理したい、などの声が多く、VulsのSaaS版で対応することになりました。

また現在、他サービスと連携したエコシステムの形成も進めています。Vuls SaaSは2017年冬頃公開予定ですので、もし興味があれば使っていただけると幸いです。

▼表1 CVSSで用いられる評価基準

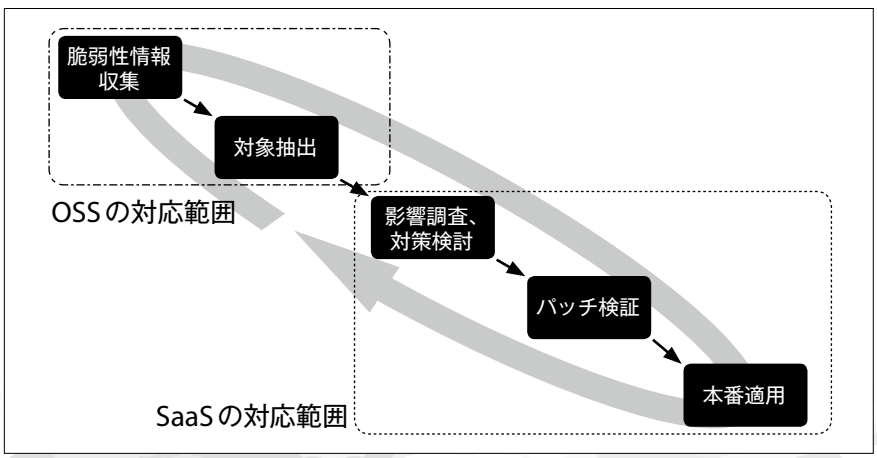
基準	説明
基本評価基準	脆弱性そのものの特性を評価する基準
現状評価基準	PoCの出現有無や対策情報が利用可能であるかといった基準
環境評価基準	製品利用者の利用環境も考慮し、最終的な脆弱性の深刻度を評価する基準



本特集をご覧になっている読者の中でも、今までセキュリティ対策をしてこなかった人、組織は多いはずです。その理由として、管理しているシステムが抱える脆弱性の深刻度を把握しておらず、「うちは大丈夫！(根拠なし)」のように、どこか他人事のように感じていたという点もあるのではないかと思います。

Vulsを使うことで、実際にシステム内に存在する脆弱性の深刻度が可視化されます。今まで対策をしてこなかった場合、恐らく多数の脆弱性が検知されます。その結果を見ればセキュリティ対策の必要性を感じられるはずですので、ぜひ一度Vulsを使って可視化してみてください。もちろん可視化で終わりではなく、継続してパッチ適用などを運用していく必要があります。その際は、前述したようなパッチを適用しやすいシステム構築などを検討していただくのが良いかと思います。 SD

▼図3 OSSとSaaSの対応範囲



ひみつのLinux通信

作)くつなりょうすけ
@ryosuke927

第44回 りなてえぴっく(Linux Team EPIC)

「SD編集部!? 爆破したはずでは……」とくつな先生の書き込みに、冷や汗が止まらない担当。「お願いです! 諦えないでください!!」(心の声……)



to be Continued

ピピーッ。おっと、もうムダなlsはもう実行させねぇぜ。宇宙人探索分散コンピューティングにでも協力したほうがいくらか有益だ。ピピーッ。おっと、もうムダなsedは実行させねぇぜ。正規表現を理解してないからTry & Errorし続けるんだ。しっかり覚えて、二度と失敗するな。ピピーッ。おっと、もうムダなスクリプトは書かせないぜ。似た機能のスクリプトばっか作りやがって。管理せずに作ってそのまま放置するから繰り返すことになるんだ。時間と資源は無駄にするな。……って新人のころに言われて泣いたことがあります。誰ですか、毎月このマンガが載ってるページが資源のムダづかいだって言ってる人は。本気で泣くぞ(被害妄想です)。

サーバーレスで実現！

素材集サービスを効率化した自動画像管理システムに学ぶ
Serverless Framework開発入門

Author
星 直史（ほし なおし）　ピクスタ㈱

Blog http://blog.naoshihoshi.com/
Twitter @NaoshiHoshi
Facebook https://www.facebook.com/naoshi.hoshi
Mail hoshinaoshi@gmail.com

話題のサーバーレスアーキテクチャですが、実際はどんなことに使えるのか、まだあまり想像できない方も多いのではないでしょうか。本記事では業務で採用されている機能をもとに、画像データを保存しサムネイル作成を行う機能を開発します。サーバーレス開発の参考に試してみてください。

はじめに

6月に開催されたAWS Dev Day Tokyo 2017[注1]のDay4では、すべてのセッションがサーバーレスに関するものでした。これは、2014年11月にAmazon Web Services（以後AWS）より発表されたAmazon Lambda（ラムダと読む。以後Lambda）をきっかけに、「サーバーレス」や「サーバーレスアーキテクチャ」に注目が集まってきているためだと思います。

筆者が所属するピクスタ㈱でも、アップロードシステムにおいてLambdaを使用して画像の処理をしており、一部の機能ではServerless Framework（詳細は後述）も使っています。AWS Dev Day Tokyo 2017 Day4の「［タワーズ・クエスト］Testable Lambda」セッションは、そのようなPIXTAシステムでのLambdaの処理を題材にしたテスト戦略についての発表でした。

弊社がLambdaを使用することで受けている恩恵は次のとおりです。

- **低コスト**
  Lambdaファンクションのリクエスト数とLambdaファンクションの実行時間により課金されるため、Amazon EC2（以後EC2）のようにアイドリング中も料金がかかることがなくなった。このためEC2で運用した場合と比較すると低コストになった
- **スケールメリット**
  毎回のトリガーに対してコードを実行させることにより自動的にスケールするため、コンピューティングリソース不足になる状況を避けることができる。また、大量アップロードがあった際もコードは並列で実行されるため、適切な規模（スケール）に常に最適化される
- **メンテナンスフリー**
  サーバのプロビジョニングや管理の必要がなくなるため、実行されるコードの実装のみに集中できる。このため運用管理コストが下がった

また、Lambdaがリリースされてから3年弱が経ち、フレームワークも数多く提供されています。なかでもServerless Framework[注2]の人気は高く、使用例やTIPSについての記事も多く存在し、Lambdaを扱ううえでのフレームワークのデファクトスタンダードになっているのではないかと思います。

この記事の趣旨と対象読者

この記事では、Serverless Frameworkがどのようなものなのかを学ぶために、画像のアップロード処理とリサイズ処理を行うシステムを構築することを

注1 https://aws.amazon.com/jp/summit2017-report/devday/

注2 https://serverless.com/

目標にします。“習うより慣れろ”の精神で、実際にコードに触れながら解説をしていきます。解説の前提は次のとおりです。

- Serverless Frameworkのバージョンは1.10.2
- Lambdaのランタイムはnode 6.10
- AWSの各サービスを使用し、連携することを前提にServerless Frameworkを使用

サンプルアプリケーションを動かそう

本記事ではServerless Frameworkを使用し、画像アップロードシステムを作成していきます（**図1**）。具体的な処理は次のとおりです。

1. Amazon API Gateway（以後API Gateway）で設定したAPIエンドポイントに画像データをPOST
2. Lambdaが画像を受けとり、Amazon S3（以後S3）に格納
3. S3にアップロードされたことをトリガーにAmazon DynamoDB（以後DynamoDB）に画像情報などを格納
4. DynamoDBに格納したことをDynamoDB Streamsを通じてサムネイル作成処理用のLambdaが起動

これらの処理を試すことで、Serverless Framework＋AWS連携でよく使われる、次のような挙動の実装方法を学ぶことができます。

- API GatewayとLambdaのつなぎこみ
- S3のイベントトリガーで起動するLambdaの設定
- DynamoDBのイベントトリガーであるDynamoDB Streamsで起動するLambdaの設定

これらはServerless Frameworkでもサポートされているベーシックな処理であるため、使用頻度は高いと思います。

Serverless Frameworkのインストールとバージョンの確認

まず初めに、Serverless Frameworkのインストールを行い、バージョン情報を確認します。

```
$ npm install serverless
$ sls -v
1.10.2
```

このような結果が得られれば正常にインストールが成功しています[注3]。

サンプルプログラムをclone

本記事では、筆者のリポジトリ[注4]にあるサンプルプログラムをもとにして解説を行います。

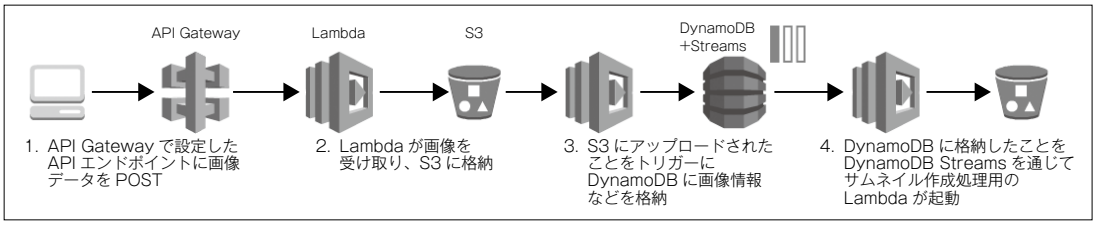
まずは`git clone`コマンドでローカルにリポジトリをcloneし、どのようなファイルが入っているか確認しましょう。

```
$ git clone https://github.com/hoshinaoshi/ServerlessSample.git
$ cd ./ServerlessSample/my-services
$ ls
```

中身のファイルはそれぞれ**表1**のように役割（処理）が分かれています。基本的に1ファイル1ファンクションになっています。また、S3トリガーとDynamoDB Streamsを使用して、upload.jsの結果からafter_upload.jsが実行され、さらにその結果からthumbnail.jsが起動されるといった処理の流れになります。そして、このトリガーの設定や、格納するバケットやテーブル情報をserverless.ymlに記述しています。

注3　slsコマンドはserverlessコマンドのエイリアスです。
注4　**https://github.com/hoshinaoshi/ServerlessSample**

▼**図1　サンプルアプリケーション概要図**



▼表1　ファイルの役割

ファイル名	役割（処理）
serverless.yml	Serverless Frameworkを使用するうえでの設定ファイル。このファイルは、どのAWSサービスを使用するか、トリガーの設定を書く
pixta.jpg	API GatewayにPOSTするための画像データ
upload.js	API Gatewayから画像ファイルを受け取り、S3に格納するLambdaファンクション
after_upload.js	upload.jsでS3に格納したイベントトリガーで起動し、DynamoDBに画像の情報を格納するLambdaファンクション
thumbnail.js	after_upload.jsで格納したDynamoDBのデータのDynamoDB Streamsをトリガーにして起動し、upload.jsでS3にアップロードされたpixta.jpgのサムネイルを作成

サンプルプログラムのデプロイと実行

サンプルプログラムは、clone直後にデプロイをしても動作するようにしています。まず最初のステップとして、画像をS3にアップロードし、画像のデータをDynamoDBに格納するところまでを説明します。

アプリケーションをデプロイするには**aws configure**コマンドでアクセスキーなどのクレデンシャル情報を設定します[注5]。

```
$ aws configure
AWS Access Key ID [********************]:
AWS Secret Access Key [********************]:
Default region name [us-west-2]:
Default output format [text]:
```

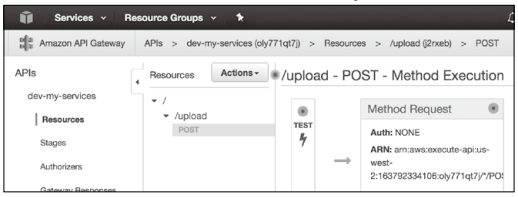
注5　aws cliのインストールは公式ドキュメントを参照。
http://docs.aws.amazon.com/ja_jp/streams/latest/dev/kinesis-tutorial-cli-installation.html

▼図2　デプロイ結果

```
naoshihoshi 07:53:47 my-services$ sls deploy -v
Serverless: Packaging service...
CloudFormation - UPDATE_COMPLETE - AWS::CloudFormation::Stack - my-services-dev
Serverless: Stack update finished...
Serverless: Removing old service versions...
Service Information
service: my-services
stage: dev
region: us-west-2
api keys:
  None
endpoints:
  POST - https://[redacted].execute-api.us-west-2.amazonaws.com/dev/upload
functions:
  upload: my-services-dev-upload
  after_upload: my-services-dev-after_upload

Stack Outputs
AfterUnderscoreuploadLambdaFunctionQualifiedArn: arn:aws:lambda:us-west-2:16379
```

▼図3　デプロイ直後のAPI Gateway



awsコマンドの設定ができたらデプロイの準備が完了です。さっそくデプロイしてみましょう。

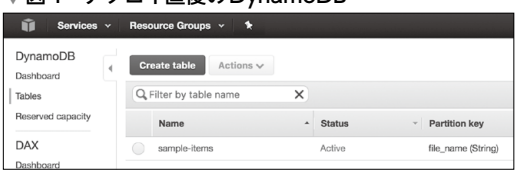
```
$ sls deploy -v
```

図2のような結果になればデプロイ成功です！作成されたAPI Gateway、S3バケット、DynamoDBテーブル、Lambdaファンクションを AWSマネジメントコンソール上で確認してみましょう。

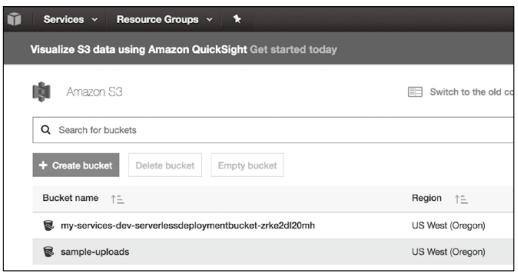
API Gatewayでは、/uploadというエンドポイントが作成されていることが（**図3**）、DynamoDBでは、sample-itemsというテーブルが作成されていることが（**図4**）、そしてS3には、sample-uploadというテーブルが作成されていることが（**図5**）それぞれ確認できるはずです。

また、**図2**の枠線で囲ったURLが、API Gatewayに作成されたAPIエンドポイントとなります（読者のみなさんは異なるAPIエンドポイントになります）。こ

▼図4　デプロイ直後のDynamoDB



▼図5　デプロイ直後のS3



のエンドポイントに画像をPOSTしてみましょう。

```
$ curl -X POST -H "file_name: pixta.jpg" -H "Accept: image/jpg" -H "Content-Type: image/jpg" --data-binary "@pixta.jpg" APIエンドポイントのURL
```

上記の処理が終わったあとにS3のsample-uploadsバケットとDynamoDBのsample-itemsテーブルを確認すると、それぞれデータが格納されていることがわかります。

serverless.ymlの解説

先述のとおり、serverless.ymlの役割は、どのAWSサービスを使用するか、トリガーの設定をどうするかなどを書く設定ファイルです。コードを見ると大きく4つのブロックがあります(**表2**)。

リスト1のように、resourcesブロックではS3のsample-thumbnailsバケットとDynamoDBのsample-itemsテーブルを生成する設定が書かれています。DynamoDBについては、indexの設定もしています。今回はファイル名をキーにするテーブルになるよう、設定しています。

S3については、サムネイル格納用のバケットの定義のみを書いています。オリジナル画像格納用のバケットについては**リスト2**のfunctionsブロックを見てください。こちらで解説します。functionsブロックは各Lambdaファンクションごとの設定を書きます。

uploadファンクションは、upload.jsのハンドラを実行し、eventsブロックにAPI Gatewayのエンドポイントを設定するための情報を書いています。今回は、画像をPOSTしたいので、methodはpost、pathはuploadとしています。これだけの記述で、API GatewayにPOSTメソッドを受け付ける/uploadというAPIエンドポイントが作成されます。

after_uploadファンクションは、画像の情報をDynamoDBに格納するファンクションです。先述

▼**リスト1　resourcesブロック(ServerlessSample/my-services/serverless.yml)**

```
...(略)...
resources:
    Resources:
        ThumbBucket:
            Type: AWS::S3::Bucket
            Properties:
                BucketName: "sample-thumbnails"
        ImagesTable:
            Type: AWS::DynamoDB::Table
            Properties:
                TableName: "sample-items"
                AttributeDefinitions:
                    -
                        AttributeName: file_name
                        AttributeType: S
                KeySchema:
                    -
                        AttributeName: file_name
                        KeyType: HASH
                ProvisionedThroughput:
                    ReadCapacityUnits: 1
                    WriteCapacityUnits: 1
```

▼**リスト2　functionsブロック(ServerlessSample/my-services/serverless.yml)**

```
...(略)...
functions:
    upload:
        handler: upload.handle
        events:
            - http:
                    path: upload
                    method: post
    after_upload:
        handler: after_upload.store_metadata_for_dynamo
        events:
            - s3:
                    bucket: "sample-uploads"
                    event: s3:ObjectCreated:*
...(略)...
```

▼**表2　serverless.ymlに書かれた各ブロックの役割**

ブロック名	役割
service	任意のサービス名を記述
provider	Lambdaファンクションを実行するための設定。ランタイムや実行環境、リージョン、Lambdaファンクションの権限を設定するブロック
functions	Lambdaファンクションごとのトリガーや REST APIの設定を記述するブロック
resources	AWS各サービス(S3のバケットやDynamoDBテーブル)のリソース生成情報を設定するブロック

のuploadファンクションはAPI Gatewayから実行される設定をしましたが、今回はS3に格納されたことをトリガーにするため、eventsブロックには、トリガーとなるS3バケットと、イベントの設定をします。このイベントはS3オブジェクトが生成された際に実行したいので、**s3:ObjectCreated:***と設定をしています。

これでS3のトリガーをもとに実行されるLambdaファンクションの設定が完了です。

リスト3のproviderブロックはLambdaファンクションが実行する環境などを定義します。ここで最も重要なのはiamRoleStatementsです。iamRoleStatementsはLambdaファンクションが実行する際のIAMRoleの設定を行います。どのResourceのActionに対して許可するのか(Effect)を定義します。今回は"*"を使用しているので、S3、DynamoDB、Lambdaに関してはどのような処理でも実行権限が与えられています。

▼リスト3　providerブロック(ServerlessSample/my-services/serverless.yml)

```
...(略)...
 3 provider:
 4     name: aws
 5     runtime: nodejs6.10
 6     stage: dev
 7     region: us-west-2
 8     iamRoleStatements:
 9         - Effect: "Allow"
10             Action:
11                 - s3:*
12                 - dynamodb:*
13                 - lambda:*
14             Resource:
15                 - "*"
...(略)...
```

●upload.jsの解説

Lambdaのお作法として、eventパラメータ、contextパラメータの受け取り設定やcallback設定ができます(**リスト4**、**表3**)。

upload.jsの役割は、API Gatewayから受け取った画像データをS3に格納するのみです(**リスト5**)。API Gatewayから渡される値は、すべてeventパラメータに格納された状態でハンドラに渡されます。また、ハンドラ内で簡単に扱うために、JSONオブジェクトにすると扱いやすくなります。

▼表3　パラメータの役割

パラメータ	役割
event	イベントデータをハンドラに渡す
context	実行中のLambda関数のランタイム情報をハンドラに渡す
callback	呼び出し元に情報を返す場合に使用

▼リスト4　upload.jsの解説(1) (ServerlessSample/my-services/upload.js)

```
...(略)...
 5 module.exports.handle = (event, context, callback) => {
 6     const json = JSON.parse(JSON.stringify(event, undefined, 1));
 7     console.log(json);
 8     console.log(JSON.stringify(context, undefined, 1));
...(略)...
```

▼リスト5　upload.jsの解説(2) (ServerlessSample/my-services/upload.js)

```
...(略)...
10     const buffer = new Buffer(json.body.replace(/^data.+,/,''), 'base64');
...(略)...
23     var bucket = new aws.S3(options);
24     bucket.putObject(params, function(err, data) {
25         if (err) {
26             console.log(err, err.stack);
27             callback(null, generateResponse(event, err.statusCode, err.message));
28         } else {
29             console.log(data);
30             callback(null, generateResponse(event, 200, 'Upload Success!!'));
31         }
32     });
...(略)...
```

bodyにはbase64エンコードされた画像データが格納されているので、それをBufferに書き込んでいます。

あとは、AWS SDKを使用し、S3に画像データをPUTするだけで、このファンクションの役目は終了です。

●after_upload.jsの解説

upload.jsと同様、お作法的にevent、context、callbackを設定します。

このファンクションの役割は、格納されたS3のオブジェクト情報をもとにDynamoDBにデータを格納するだけです。upload.jsと異なるのはeventパラメータに格納されているデータです。このハンドラは、s3:ObjectCreatedトリガーをもとにしていますので、**リスト6**のようなeventを受け取ります。このeventパラメータの情報をDynamoDBに永続化することで、さまざまな用途に使用することが可能になります。

次にDynamoDBへのPUTを行います(**リスト7**)。DynamoDBを操作するAWS SDKをrequireし、テーブル情報とそのテーブルに格納する情報を設定して、putItemするだけとなります。唯一、注意が必要なところは、テーブルにおいて数値型であるsizeカラムへ情報を書き込む際、値は文字列型にしなければならないことです。数値型のカラムに対して数値を渡すとエラーになってしまうのです。

サムネイル作成処理

画像アップロード処理の次は、アップロードされた画像をもとにサムネイルを作成する処理を作成します。先述のとおり、DynamoDB Streamsをトリ

COLUMN

Lambdaファンクションの権限

今回用意したserverless.ymlのiamRoleStatementsブロックは、すべてのS3、DynamoDB、Lambdaのアクションを許可しています。この設定は記事向けに、“まずはServerless Frameworkを動かしてみること”に主眼を置いたため、あえてこの設定にしています。本来は意図しないデータソースへの意図しない操作を行わないために、最小限の権限設定をします。また、AWS Cognitoを使用する場合は、CognitoのIDごとのリソース許可にしなければ、ほかのユーザのデータソースを更新／削除してしまうリスクがあります。そのため、IAMの権限レベルで制御したほうが、より堅牢なシステムにすることができます。

▼リスト6　s3:ObjectCreatedトリガーで起動したLambdaに渡されたevent引数の内容

```
{
   "Records": [
   ...（略）...
           "s3": {
               "s3SchemaVersion": "1.0",
               "configurationId": "3ac4ad41-7a79-43b3-ae04-0e5fb0f1ecc0",
               "bucket": {
                   "name": "sample-uploads",
                   "ownerIdentity": {
                       "principalId": "A2QJEYVO50AIW2"
                   },
                   "arn": "arn:aws:s3:::sample-uploads"
               },
               "object": {
                   "key": "pixta.jpg",
                   "size": 41926,
                   "eTag": "8ea4fbcefef770577111ca2178c25092",
                   "sequencer": "00597683211D75B33B"
               }
           }
       }
   ]
}
```

▼リスト7　after_upload.jsの解説 (ServerlessSample/my-services/after_upload.js)

```
...（略）...
    event.Records.forEach(function(record) {
        const params = {
            TableName: 'sample-items',
            Item: {
                'bucket':           { 'S': record.s3.bucket.name },
                'file_name':     { 'S': record.s3.object.key },
                'size':              { 'N': record.s3.object.size.toString() },
                'uploaded_at': { 'N': (new Date()).getTime().toString() }
            }
        };
        dynamoDB.putItem(params, function (err, data) {
            if (err) {
                console.log(err, err.stack);
                callback(null, generateResponse(event, err.statusCode, err.message));
            } else {
                console.log(data);
                callback(null, generateResponse(event, 200, 'Upload Success!!'));
            }
        });
    });
...（略）...
```

ガーにLambdaファンクションを起動させるように対応します。

●DynamoDB Streamsとは

DynamoDB Streamsとは、DynamoDBに対するすべての更新（Put、Update、Delete）が24時間保持される機能です。また、テーブル更新などのイベントをフックにすることもできるので、Amazon SNS通知など、さまざまな処理を行えます。今回はイベントをフックし、Lambdaを動かしていきます。

●DynamoDB Streamsの設定

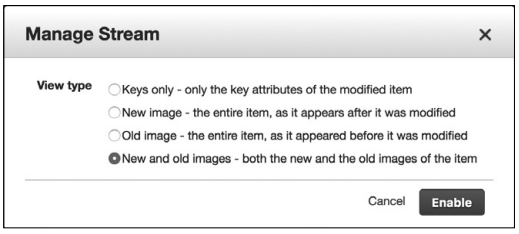
バージョン1.10.2時点のServerless Frameworkでは、DynamoDB Streamsを使用してLambdaを実行する設定はできますが、DynamoDBのテーブルに対して、DynamoDB Streamsの設定をすることができません。そのためDynamoDB StreamsはAWSマネジメントコンソールから手動で有効化する必要があります。

AWSマネジメントコンソールから「DynamoDB」→「Tables」→「テーブル（sample-items）」を選択し、「Overview」→「Stream details」→「ManageStream」を選択します。すると**図6**のように、ストリームを経由して渡されるデータ構造を選択できるラジオボタンが表示されます。 それぞれ挙動は**表4**のとおりです。

「New and old images」を選択した場合が最も情報量が多く、「Keys only」が最も少なくなります。データの内容に変わりはないので、今回はわかりやすさと実際にどんなデータが入ってくるのか確認する目的のもと、「New and old images」を選択します。

設定が終わるとDynamoDB Streamsが有効になり、テーブルの「Overview」タブ→「Stream details」

▼図6　DynamoDB Streamsの設定



▼表4　各ラジオボタンの挙動

設定名	挙動
Keys only	変更された項目のキー属性のみ取得
New image	変更後のデータのみ取得
Old image	変更前のデータのみ取得
New and old images	変更後、変更前のデータの両方取得

に「Latest stream ARN」が表示されます。このARNはserverless.ymlに設定する必要があるので、メモしておきましょう。

●serverless.ymlの設定

リスト8のように、serverless.ymlのfunctionsブロックの中にthumbnailファンクションの記述がコメントアウトされています。このコメントアウトを解除し、arnの個所を先ほどメモしたDynamoDB StreamsのARNに変更します。このevents設定は、指定したARNのDynamoDB Streamsを使用し、Lambdaファンクションを起動させるための設定です。

●thumbnail.jsの解説

thumbnail.jsはサムネイルを作成する役割のLambdaファンクションです。サムネイルを作成するためにはリサイズ用のモジュールを別途ロードする必要があります。今回はgm（GraphicsMagick[注6]）を使用します。

まず初めに、npmでgmをインストールします。cloneしたフォルダ内にpackage.jsonがありますので、`npm install`を実行します。インストールが終わると、node_modulesフォルダが作成され、gmのインストールが完了です。

```
$ npm install
$ ls
after_upload.js         node_modules
package.json            pixta.jpg
serverless.yml          thumbnail.js
upload.js
```

注6　http://www.graphicsmagick.org/

COLUMN

Lambdaのスペック

LambdaのCPU能力はメモリの設定に比例します。serverless.ymlのfuctionsブロックに記述してあるLambdaファンクションごとに「memorySize」を指定することができます。処理に時間がかかっている場合は、memorySizeを変更することで、処理が高速化される場合があります。

thumbnail.jsでは、これまでと同様にS3と連携する必要があるため、AWS SDKもロードします（**リスト9**）。

gmの使用方法については誌面の都合上割愛しますが、公式のドキュメントがたいへん参考になります[注7]。14行目からサムネイル作成の処理が開始されますが、Lambdaではファイル入出力が難しいため、ストリームとバッファに書き込み処理をしていきます[注8]。

最後にS3 sample-thumbnailsバケットにPUTしてサムネイル作成が完了です。

修正が終わったあとは、変更を反映させるため、再度デプロイをします。

```
$ sls deploy -v
```

サンプルアプリケーションの動作確認

デプロイが完了したら、いよいよ実際に動くかを

注7　http://aheckmann.github.io/gm/docs.html
注8　容量が大きい画像を処理する場合は、メモリが必要です。

▼リスト8　serverless.ymlのthumbnailファンクション部分（ServerlessSample/my-services/serverless.yml）

```
... (略) ...
 29 #    thumbnail:
 30 #        handler: thumbnail.create
 31 #        events:
 32 #            - stream:
 33 #                  arn: "arn:aws:dynamodb:us-west-2:163792334106:table/sample-items/stream/2017-
07-17T16:13:40.753"
 34 #                  batchSize: 1
 35 #                  startingPosition: TRIM_HORIZON
 36 #                  enabled: true
... (略) ...
```

▼リスト9　thumbnail.jsの解説 (ServerlessSample/my-services/thumbnail.js)

```
...（略）...
 3 const gm = require('gm').subClass({ imageMagick: true });
 4 const s3 = new AWS.S3();
...（略）...
13
14     s3.getObject({Bucket: originalBucket, Key: objectKey}, function(err, data) {
15         // 画像処理
16         gm(data.Body)
17         .resize('450', '350')
18         .borderColor('gray')
19         .border('245', '245')
20         .gravity('Center')
21         .crop('450', '350')
22         .stream(function(err,stdout,stderr) { // ストリームで出力する
...（略）...
42                 s3.putObject(params, function(err, data) { // S3に書き出す
43                     if(err) {
44                         console.log(err, err.stack);
45                         callback(null, generateResponse(event, err.statusCode, err.message));
46                     } else {
47                         console.log(data);
48                         callback(null, generateResponse(event, 200, 'Upload Success!!'));
49                     }
50                 });
51             });
...（略）...
57 };
...（略）...
```

確認します。

```
$ curl -X POST -H "file_name: pixta.jpg" -H "Accept: image/jpg" -H "Content-Type: image/jpg" --data-binary "@pixta.jpg" APIエンドポイントのURL
```

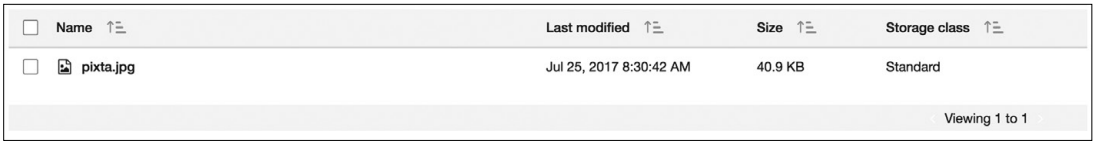
API Gatewayのエンドポイントに向けてPOSTが終わったら、今度は作成されたサムネイルが正常にできているかを確認します。

AWSマネジメントコンソールから「S3」→「sample-thumbnails」→「thumb_pixta.jpg」として上記のファイルを確認してみると、グレーの背景でリサイズされた画像が表示されます。また、画像のサイズもオリジナルと比較すると小さくなっているので、適切にリサイズされサムネイル化ができていることがわかります（**図7～9**）。

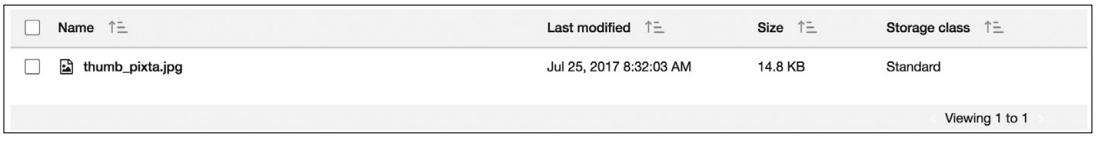
まとめ

Serverless Frameworkを触ってみていかがでしたでしょうか。サンプルアプリケーションですの

▼図7　s3://sample-uploadsバケットのオリジナル画像



▼図8　s3://sample-thumbnailsバケットのサムネイル画像



▼図9　作成されたサムネイル画像



で、規模が小さいとはいえ、リポジトリをcloneしてからdeploy、動作させるまでのハードルがとても低かったのではないでしょうか。serverless.ymlに記述した構成やトリガーはCloudFormationで実行されるため、冪等性（べきとうせい）が担保され、その結果、すぐに動作する環境が整うのだと思います。また、EC2やRDSの運用とは異なり、金銭的なコストもとても低いです。このサンプルアプリケーションを1ヵ月そのまま放置したとしても、1ヵ月に1ドルもかからないと思います[注9]。

また、Serverless FrameworkではAPI Gateway、Lambda、DynamoDBを組み合わせることで、高いスケーラビリティを得られます。試しに筆者が同時に1,000枚の画像アップロードしたところ、1枚あたりの実行完了時間を維持したまま、すべての画像を処理しきりました。EC2やRDSを使用した場合、スケーラビリティ、パフォーマンス、コストなど、常に多くのことを管理しなければならないため、管理コストが段違いなのではないかと思います。

一方、Serverless Framework単体で使用した場合の問題点もいくつかあります。

本記事に書いた内容でいくと、DynamoDB Streamsの有効化は、Serverless Frameworkから設定変更することができず、AWSマネジメントコンソールから有効化し、そのARNをserverless.ymlに書くことになります。そのため、Serverless Frameworkの枠組みから外れることになり、運用を想定した場合、何かしらの方法でその手順を共有する必要がでてきます。

また、実際にデプロイしないと動作させられないことも扱いにくい点ではあります。Serverless Framework単体で動作確認しようとすると、実装後やデプロイ後に、`sls logs`コマンドやCloud WatchLogsで確認するしか方法がありません。確認する方法はあるのですが、実装段階であっても外部サービスを通じてログを見て作業することは効率が悪いです。この解決方法として、冒頭に記載した「[タワーズ・クエスト]Testable Lambda」セッションで紹介されたLocalStack[注10]を使用することや、自動テストによる動作確認環境を整えることで回避することは可能です。

最後はS3トリガーやDynamoDB Streamsを使用することで、処理が複雑化することが懸念されます。今回は小規模のサンプルアプリケーションであるため、まだ追うことは可能ですが、さらにAWS SQSやAWS Kinesisストリームを使用したことを考えた場合、「結局このLambdaは何をもとに起動されるのか？」が追えなくなってしまう懸念があります。

弊社では、画像処理システムにLambdaを使用していたり、一部のcron処理でServerless Frameworkを使用しています。サーバーレスアーキテクチャは、そのほかにもSPA（シングルページアプリケーション）のバックエンドや、監視、運用、保守の運用業務や、ログやユーザの行動をストリームで処理することなど、さまざまな用途で使用することが考えられます。

本記事ではServerless Frameworkの基本的な動きとその設定を紹介しました。ほかにもAWS SNS、Kinesis、Cognito、IAMなど、Serverless Frameworkで扱えるAWSサービスは多く、これらを組み合わせることで多様なシステムの実装が可能になります。またAWSのほかにも、Google Cloud PlatformやMicrosoft Azureなどのパブリッククラウドサービスに対応しているため、システムの実現方法や選択肢がとても広がるのではないかと思います。この記事を通して、Serverless Frameworkを適用させる場面に少しでも寄与できれば幸いです。SD

注9　API Gatewayは全世界に公開されている状態であるため、一通り触り終えたら「sls remove」コマンドで環境を削除することを強くお勧めします。

注10　https://github.com/atlassian/localstack

特別企画

ネットワークエンジニア その技術の極め方

Author 釜山 公徳(かまやま まさのり)
日本電気㈱金融システム開発本部
金融デジタルイノベーション技術開発室
Twitter @masart_3
Mail mklnx238@mklnx.sakura.ne.jp

ネットワークエンジニアに求められる変化



マルチスキルになろう！

ITインフラにはネットワークを含むさまざまな要素があります。ネットワークエンジニアは、スイッチング／ルーティングといったネットワーキング技術に特化したプロフェッショナルです。サーバとエンドポイントとの道を作り、その経路を保全することがミッションです。それゆえネットワーク機器単体だけを考えるのではなく、全体を考える必要があります。つまり「木を見ず、森を見る」です。しかしながら「ここまではネットワーク側の問題だけど、ここからはサーバ側の問題」といったサーバエンジニアとネットワークエンジニアの行き違いがたびたび発生するのが現実です。その結果、構築スケジュールやインシデント対応などで余計に時間を費やし、遅延が発生することも珍しくありません。そうした背景にはさまざまな問題がありますが、その解決策として双方の技術領域を理解した「マルチスキルを有するエンジニア」が求められていると筆者は考えます。

これから必要な技術とは何か

これまではオンプレミスあるいはISP側だけを考えればよかったのですが、世界規模でクラウド化が進んだ結果、プライベート／パブリック／ハイブリッドクラウドのすべてを含めたネットワークを考える必要が出てきました。この範囲が今まさにネットワークエンジニアに求められています。クラウドに応じた基礎設計ができることは言うまでありません。さらにクラウドには次の特性があります。

・従量課金
・ネットワークトラフィックの増大
・処理が通信ベース

そのためQoS(Quality of Service：ネットワークの品質)やフロー制御の設計、ピーク時のトラフィック算出をより厳密に行わねばならなくなりました。ネットワーク技術を単純に身につけているだけでは、ビジネスの現場で必要とされなくなります。

技術を点ではなく線や面でとらえよ



パケットダンプ解析(以降、ダンプ)ができないとネットワークエンジニアとは言えない――

筆者プロフィール

PROFILE

サーバ・ネットワークのオールフェーズ(開発、設計、構築、運用)、テクニカルサポート、セキュリティコンサルティング、マルウェア解析などを経て、現在先端技術によるセキュリティプランニングに従事。一度目の大学では法律学、二度目は電気工学、専門学校にてファッションデザインを専攻。休日はおもにギターやベースの演奏、電子工作、シャドーボクシング、ファッションデザイン画に取り組む。バーボンウヰスキー、赤ワイン、ラムチョップをこよなく愛する。

という方がよくいますが、もはや正しくありません。ダンプはネットワークを流れる詳細な情報の点です。その点と点とをつなぎ、通信フローを紐解いていけば、個々のネットワークを解析していると言えます。たしかに、その観点では間違いではありませんが不十分な解答です。ダンプは基本的にミクロな視点です。今は、ネットワーク全体での構成やトラフィックフロー、強いてはルーティングやスイッチング、つまりマクロな視点が必要とされています。ネットワークエンジニアに求められる基礎技能は、マクロにネットワークを理解することです。その技術をピラミッド的な階層にたとえると、下層にマクロ的な技術があり上層にミクロ的な技術があると考えるとわかりやすいでしょう。

パケットの構造を理解することは基本です。ただしそれは「点の情報」でしかありません。その前にネットワーク全体を把握すること、つまり技術や知識を点ではなく線、さらに面で見ることをお勧めします。

そもそも、ネットワークとはノード同士のつながりを単位としているのですから――本当に優秀なエンジニアを目指すなら――点の技術に注力する前に、もっと高い視点で臨むべきです。たとえば、スタティックルーティングとダイナミックルーティングのメトリック差分についてアルゴリズムを理解することは言うまでもありませんが、ダイナミックルーティングにおけるOSPF(Open Shortest Path First)などのリンクステート型ルーティングプロトコルやRIP(Routing Information Protocol)などのディスタンスベクタ型ルーティングプロトコルの特性や、HSRP(Hot Stanby Router Protocol)やVRRP(Virtual Router Redundancy Protocol)といった冗長化プロトコルの習得まで考えるべきです。これにより、たとえばネットワーク障害が発生した場合に「ネットワーク断」が発生しても何秒から何秒の幅であるかの推算ができるようになります。クリティカルな状況で包括的かつ見通しの良い技術で迅速な対応ができる――これが優秀なネットワークエンジニアの条件です。点を見ているだけではこの対応はできません。技術を磨くヒントは、ネットワークを「面」としてとらえることです。

一歩先を行くネットワークエンジニアになる方法



何ができればいいのか？

今は設計、構築、運用のあらゆるフェーズで即座にネットワークのイメージができるエンジニアが求められています。理論の構築を積み重ね、問題解決に対して想像力を高めていくことが重要です。ほかのエンジニアにも言えますが、ネットワークエンジニアの場合は、いわゆるサーバエンジニアと比較すると次のような課題が生まれやすい状況にあります。

・評価環境の入手が困難
・最小単位の動作検証でも複数機器が必要

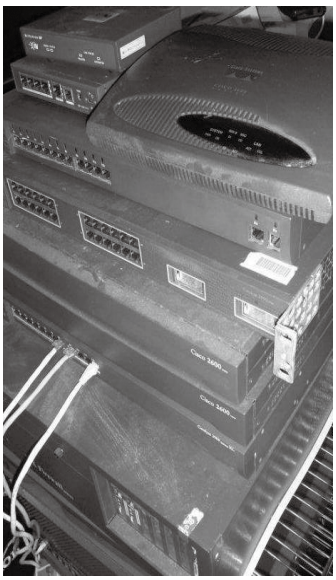
一歩先を行くための技 その1 自宅実験環境作り

とはいえ実機や環境がないと、ネットワークはなかなかイメージしにくいものです。そこで現場で使える実務的なスキルを修得するために、筆者が実践したのは「自宅でネットワーク環境を構築し検証する」ことです。筆者お勧めの機器構成は次のようになります。

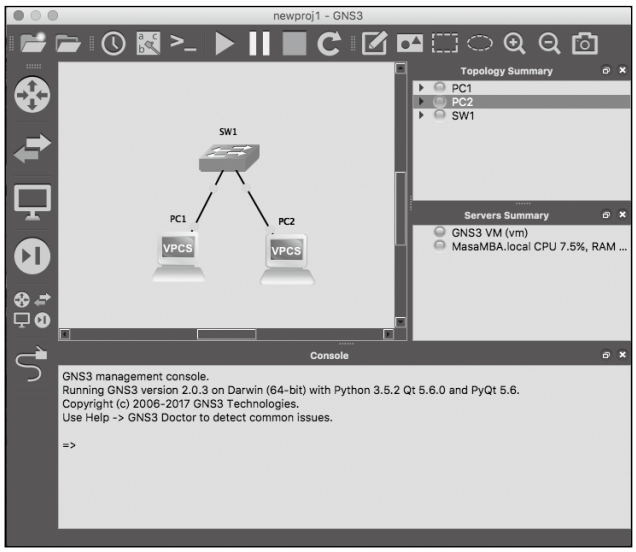
・L2スイッチ …… 2台以上
・ルータ …… 2台以上
・L3スイッチ …… 1台以上
・ファイアウォール … 1台以上

新品ならば、これらの機材を購入するには数十万から数百万円かかってしまいます。個人では難しいですが、今は中古専門店や秋葉原のジャンクショップ、そしてネットオークションがあります。筆者はそれらを活用し、1台あたり数千円から1万円程度で買いそろえることができました(**写真1**)。

▼写真1　筆者が若かりし日の実験環境



▼図1　GNS3の実行画面（https://www.gns3.com/からダウンロード可能）



実際に環境を設計・構築する際の注意すべき点としては、本番環境とトラフィックフローを意識し、冗長化構成を組むことです。さらに具体的に意識することは、コア層、ディストリビューション層、アクセス層の三階層からなるネットワークモデルです。スイッチやルータを単体で検証することは悪いことではありませんが、ネットワークの本質を理解するには不十分です。実際に構築をしてみると、基本設計、詳細設計の各フェーズのイメージがつきやすくなります。さらにデータセンターなどでサーバやネットワークをキッティングするとき、機器を容易に設置できるようになります。つまり実務的な技術を迅速に修得できるようになるメリットがあります。

一歩先を行くための技 その2
自宅ネットで障害を発生させてみる

みなさんは故意に障害を発生させたことがありますか？　物理障害は火事などの危険性があるので避けたほうがいいですが、自宅環境ならば論理障害を発生させても多少のことは問題ないでしょう。開発・本番環境にかかわらずエンジニアならば一度は経験したことがあるブロードキャストストームなどのネットワークループ障害があります。これを故意に発生させて、その動作を見てみましょう。どうすれば障害が発生し、どうすれば一次対処できるのか、そしてどう未然に防ぐかといったトラブルシューティングスキルを得る手がかりになります（ブロードキャストストームによるLEDの点滅は花火のようできれいですし、一度見てみるとおもしろいかもしれません）。そのほか、冗長構成を組んでいるルータの主系側の電源を落とし、経路変更までどれくらい時間がかかるかを実測してみるのもお勧めです。実際のネットワーク障害に対応したスキルが身につきます。

一歩先を行くための技 その3
仮想環境の活用

より柔軟に障害の検証をするために、無償で利用可能な仮想環境を紹介します。GNS3（Graphical Network Simulator-3）というツールです。CISCO Systems社、Juniper Networks社、Checkpoint社の仮想ネットワークアプライアンスや、そのほかOSSの仮想ブリッジや仮想マシンを使って、Windows/macOS/各種Linux環境などでネットワークのエミュレーションが

できます(**図1**)。

動作に多少不安定なところや、VirtualBoxやVMware社製品との連携でハードルが高い部分がありますが、仮想的にネットワーク環境を設計・構築できます。気軽に検証するには十分です。まずは実機検証をして機器に慣れ、そのあとにGNS3を利用することをお勧めします。

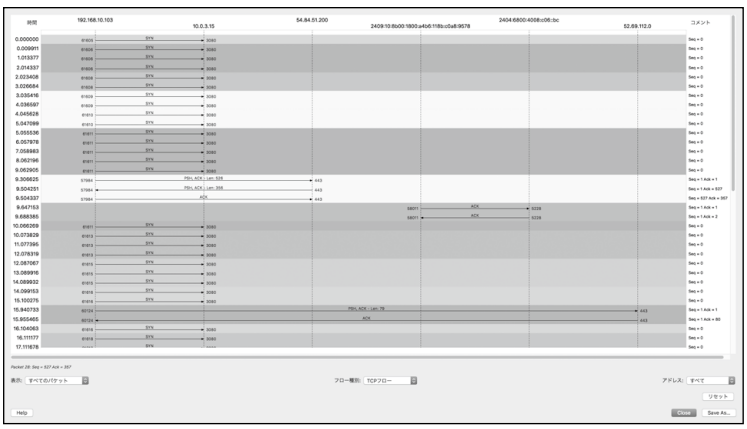
一歩先を行くための技 その4 Wiresharkの効果的な使い方

ネットワークエンジニア御用達のパケットキャプチャツール／パケットアナライザと言えば「Wireshark(https://www.wireshark.org/)」です。ネットワーク障害が発生した際、何万、何十万とあるパケットダンプから、フィルタリング機能を駆使して通信フローのどこの部分で障害が発生したかを分析・解析するために多くのエンジニアが使用しています(懐かしいツールですがその前身のEtherealもありました)。WiresharkはWindows、macOS、各種Linuxなど幅広く利用できるGUIツールであり、多機能かつ高機能であるため、これがあれば基本的には問題ありません。

Microsoftのツールか商用ツールでないと客先に導入できない場合は、Microsoft Message Analyzer[注1]やnetshコマンドを使用します。各種Linux環境ならばランレベルが3(CUIでのマルチユーザモード)の場合にtcpdumpコマンドを利用します。

Wiresharkの効果的な利用方法は、グラフ機能の利用です(**図2**)。グラフ機能は視覚的に往復遅延時間、スループットなどを出力でき、ネットワークエンジニアでない方へプレゼンテーションする場面などで活躍します。フローグラフ機能を利用すれば、通信フローを視覚化でき、障害対応における原因分析の一助になります。また、このほかにもさまざまな機能がたくさん実装されているので、状況に応じて利用すると良いでしょう。

▼図2　Wiresharkのグラフ機能実行画面



一歩先を行くための技 その5 マルウェア解析ならば「Fiddler」

このほかに簡易解析ツールとしてマルウェア解析などでよく利用されるFiddlerがあります(**図3**)。Wiresharkは高機能な反面、パケットをすべてキャプチャすることからデータ量が多く、また利用するにあたり学習コストがかかります。Fiddlerならばより簡単に利用できます。不正な通信先を確認するといった用途にはぴったりです。

仮想化が進む今のネットワーク、これから使える技術とは



サイバー攻撃とフォレンジック

昨今、世界各国で標的型サイバー攻撃の被害が多発しており、日々多くの企業でそのセキュリティインシデントの原因究明と対策に追われています。その原因究明と対策のために求められている技術を総称して「フォレンジック」と言います。ネットワークフォレンジックの場合は、デジタルトラフィック分析／捜査になります。

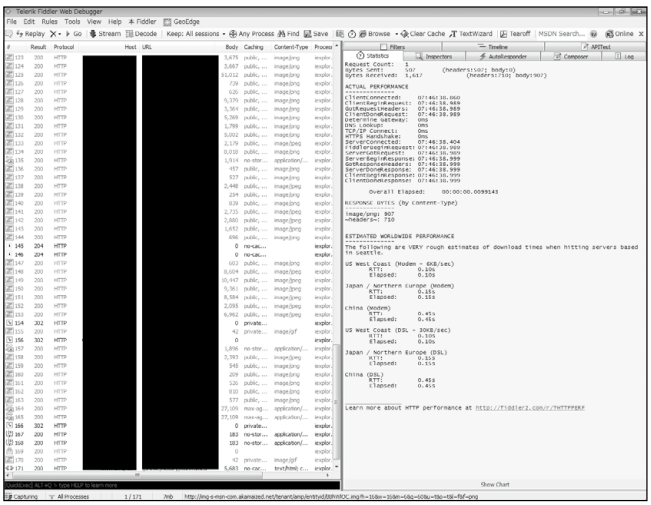
注1) URL https://www.microsoft.com/en-us/download/details.aspx?id=44226

これはダンプから不正な通信を割り出し、被害範囲を見積もります。このほかにも、セキュリティ要件は多く存在します。サイバー攻撃の場合は次になります。

・ARPスプーフィング
・ポートスキャン
・セッションハイジャック
・DoS(DDoS)攻撃

たとえばポートスキャンならばnmapを使用し、事前調査としてポートの解放状況などを確認します。また、nmapはコマンドラインツールだけでなくGUIツールもあります。ペネトレーションテストでもよく使用されます。このnmapのフォレンジックとしてWiresharkを用いた場合、**図2**のように確認できます。

▼**図3**　Fiddler(http://www.telerik.com/fiddler)



ネットワークフォレンジック

実際に現場で、リアルタイムでのWiresharkによるフォレンジックは現実的ではありません。IDS/IPSなどのネットワークアプライアンス製品や、常時エンドポイントからフォレンジックデータを収集することを目的としているEDR(Endpoint Detection and Response)製品でデータ収集します。それらをもとにセキュリティインシデントの解析を行います。なお、EDRでは次のステップで情報を収集します。

①ログ収集
②脅威の検知
③脅威への対処

仮に偵察活動を行うマルウェアが存在する場合、EDRによる解析が主流になってきたので、これを捜査するスキルも求められるようになりました。

マルウェアによる不正通信のフォレンジック

端末におけるデータフォレンジックやアンチウイルスソフトウェアによる検知、IDS/IPS(不正侵入検知・防御システム)ツールによる検知などが発生した場合、マルウェアを解析することがあります。

ネットワークの観点では、不正通信による情報漏洩やC&C(Command & Control)サーバへの通信などが懸念されます。これには前述したFiddlerが有用です。

図3のように、通信が発生するとわかりやすく出力されます。詳細な解析を要する場合はWiresharkが向いていますが、簡単に動的解析がしたいといった場合にはよく利用されます。

ネットワークエンジニアを極めるには



企業や団体によってネットワークエンジニアが担う範囲は異なるものの、規模を問わず設計・構築したり、ネットワーク上で発生するすべてのインシデントに対応できるネットワークエンジニアが、グローバルで求められる優秀な人材と言えます。目の前にある業務をこなすだけでなく、自分で一から環境を作るなどして様々な技術をキャッチアップし、ネットワークを自身の掌で転がし、掌握できるようになってネットワークエンジニアを極めましょう。SD



TensorFlow機械学習クックブック

Nick McClure 著、株式会社クイープ 訳
B5変型判／392ページ
4,200＋税
インプレス
ISBN＝978-4-295-00200-0

本書は、機械学習を多少なりとも勉強しており、Pythonプログラミングもある程度できる方が、TensorFlow（以降TF）を使った機械学習／深層学習プログラミングの基礎を身につけるためのもの。TFのしくみから線形回帰、サポートベクトルマシン、最近傍法、ニューラルネットワーク（以降NN）、自然言語処理、畳み込みNN、リカレントNNの実装が解説される。本書のポイントは、解説を「準備」「方法」「仕組み」「ステップアップ」という段階に分けて行っていること。TFはデータの扱い方や処理の概念が特徴的なのだが、この説明の仕方が理解の助けになってくれるだろう。サンプルを利用しながら試行錯誤することで、TFを使った機械学習／深層学習を自身の目的に応用する素地ができあがるはずだ。

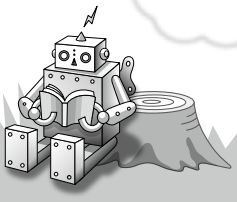


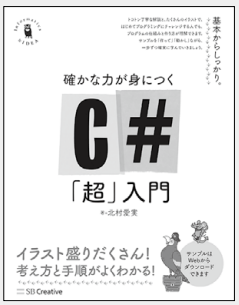
暗号技術のすべて

IPUSIRON 著
A5判／720ページ
3,500円＋税
翔泳社
ISBN＝978-4-7981-4881-6

「暗号技術のすべて」とタイトルにあるとおり、ユリウス・カエサルが考案したとされるシーザー暗号といった古典的な暗号から、公開鍵暗号といった現代の暗号、ブロックチェーンなどの暗号技術を活用した最新のセキュリティ技術まで、広く紹介している。それでいてひとつひとつの暗号技術のトピックは、単なる概要にとどまらず、「入力」「出力」「動作」というアルゴリズムの観点から、適宜数式も使って解説している。ここ数年、OpenSSLの脆弱性に始まり、直近ではGitの脆弱性などもあって、暗号・セキュリティ技術への警戒心が高まっている。本書では暗号技術に対する攻撃も併せて紹介されており、自分の環境のOSやソフトの内部でどのようにセキュリティが担保されているのかを意識しながら、自分事として学ぶことができる。

SD BOOK REVIEW





確かな力が身につくC#「超」入門

北村 愛実 著
B5変型判／400ページ
2,580円＋税
SBクリエイティブ
ISBN＝978-4-7973-9026-1

.NET Frameworkのクロスプラットフォーム版「.NET Core」の発表、Visual Studioの無償版「Visual Studio Community」の機能強化など、.NETの開発が始めやすくなっている。本書は.NET Frameworkでの開発には最適と言われるC#の入門書で、フルカラーのイラストとスクリーンショットを多用した、プログラミング初心者にも優しい作りになっている。小さなサンプルプログラムを作りながらC#の文法とオブジェクト指向、Visual C#の使い方を学んだあとは、電話帳アプリや天気予報アプリといったGUIを持ったデスクトップのWindowsアプリケーションの開発に移る。Windows Serverへのアプリの配置や配信などは扱っていないので、パソコン1台のみで気軽にハンズオンを進められる。



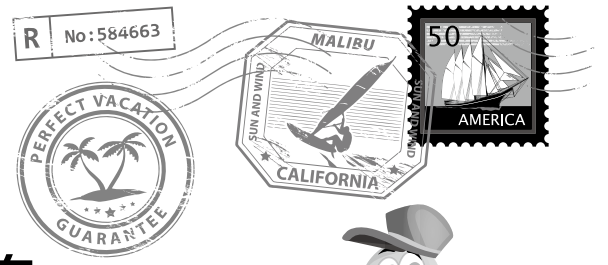
Mackerelサーバ監視［実践］入門

井上 大輔、粕谷 大輔、杉山 広通、田中 慎司、坪内 佑樹、松木 雅幸 著
A5判／320ページ
2,900円＋税
技術評論社
ISBN＝978-4-7741-9213-0

Mackerelは、㈱はてなが開発したサーバ監視のためのクラウドサービス。見やすいUIとハッカブルな作りが特徴だ。もとははてなの社内システムで、「はてなブログ」「はてなブックマーク」といった巨大Webサービスのインフラを支えてきた技術だ。本書はそのMackerelの入門書だが、はてなが今まで培ってきたインフラ管理のノウハウも併せて学ぶことができるようになっている。「メモリやCPU負荷、ネットワーク帯域量といったOSのメトリック値には、それぞれどんな性質があり、どのような観点で監視すれば良いのか」「Webアプリケーションには、どのようなプロダクトを使ってどのような構成でインフラを構成すれば、冗長性を保ちつつ監視もしやすくなるのか」など、社内のインフラ管理にすぐに応用できる知識が目白押しだ。

telnetを武器に
システムの本質を理解する

身近なメールで使われている技術をネットワークコマンドで体験してみよう





Author 中井 勘介(なかい かんすけ)
㈱構造計画研究所 SendGridエバンジェリスト
Twitter @nakansuke

世界で初めてE-mailの送信に成功したのが1971年[注1]。それから45年以上が経過していますがまだまだメールは使われ続けています。チャットやメッセージングツールの登場により、人と人とのコミュニケーションツールとして使われる頻度は減りましたが、現在でも廃れることなく使われています。Webサービスにサインアップする際に求められるのは何でしょうか。そう、メールアドレスです。

まだまだ使われ続けているE-mailでどのような技術が使われているのか、今回はネットワークコマンドを使って体験してみましょう。

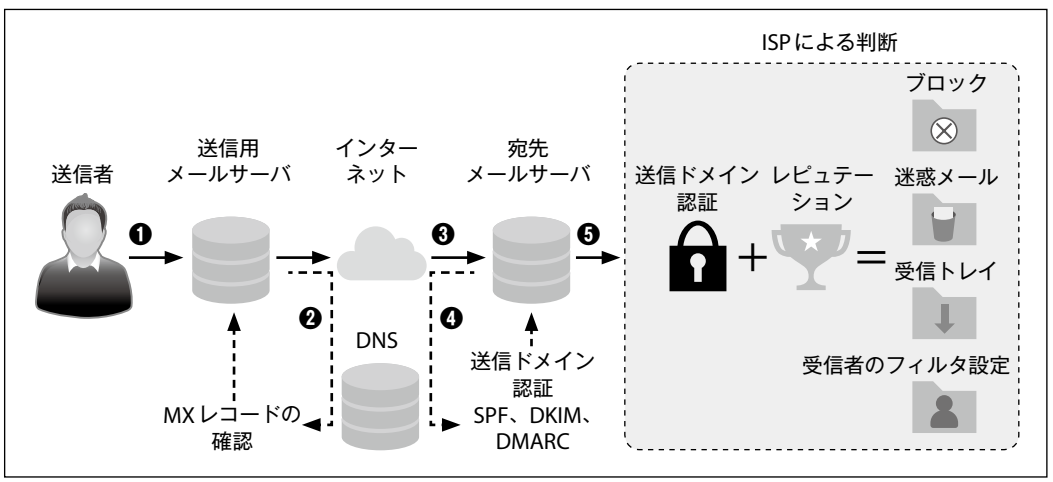


メールが届くまで

図1では、メール送信から宛先に到達するまでのフローを示しています[注2]。以降では❶SMTP通信と❹送信ドメイン認証を取り上げます。各種Telnetを使いクライアントを用意して試してみてください。

❶送信者が送信用メールサーバ(SMTPサーバ)に送信リクエスト
❷宛先ドメインのMXレコードから宛先メールサーバを確認

▼図1　送信から宛先到達までの流れ



注1) URL http://www.computinghistory.org.uk/det/6116/first-e-mail-sent-by-ray-tomlinson/
注2) 実際には各メールサーバの設定やポリシーによって細かい挙動は異なります。



▼表1　代表的なSMTPコマンド

コマンド	パラメータ	意味
HELO/EHLO	ホスト名	サーバに対し自身の身元を示す。EHLOはHELOの拡張
MAIL	FROM：＜メールアドレス＞	送信元メールアドレスを通知する
RCPT	TO：＜メールアドレス＞	宛先メールアドレスを通知する
DATA	なし	これ以降「ドット」+「改行」まではメール本文
QUIT	なし	SMTPの接続を終了する

❸宛先メールサーバに対して送信リクエスト
❹宛先メールサーバにて送信ドメイン認証を実施
❺認証結果およびレピュテーション[注3]によってメールを受け付けるか、どこに配送するかを判断



SMTPの動きを深く理解する

メールを送信するには、SMTP(Simple Mail Transfer Protocol)を使用します。普段とくに意識することはないと思いますが、Thunderbirdなどのメールクライアントアプリケーションや、GmailやHotmailなどのWebメールを使用してメールを送信する際、それらの裏側ではSMTPを利用して送信メールサーバに対するリクエストが処理されています。

SMTPコマンドのあやつり方

SMTPコマンドは、「コマンド」+「空白」+「パラメータ」+「改行」という形式で成り立ち、**表1**のようなコマンドがあります[注4]。

SMTPレスポンス

先のコマンドに対し、サーバは「応答コード」+「空白」+「フレーズ」の形式でレスポンスを返します(**表2**)。

Telnetを使いメールサーバへの送信リクエストを実行してみる

SMTPはシンプルなプロトコルで、Telnetを使用してメールサーバとやりとりをすることができます。実際に試してみましょう。今回の例ではメール配信SaaSであるSendGrid[注5]のSMTPサーバを利用します。実際に試す際には普段利用しているメールサーバ、もしくはSendGridのようなメールサービスなどを使用ください。

▼表2　代表的なSMTPレスポンス

ステータスコード	意味
2xx	コマンド成功
3xx	コマンドは受け付けられ、続きのデータを要求
4xx	一時的なエラー
5xx	恒久的なエラー

〒 Step1「接続」

次のサーバにtelnetを使い、接続してみます。

・サーバ名：smtp.sendgrid.net
・ポート番号：587[注6]

```
$ telnet smtp.sendgrid.net 587 Enter
Trying 203.0.113.0... ←接続先メールサーバのIPアドレス
Connected to smtp.sendgrid.net.
Escape character is '^]'.
220 SG ESMTP service ready at ismtpdxxxx
p1mdw1.sendgrid.net
```

〒 Step2「挨拶」

正常に接続できたので、EHLOコマンドを発行し、自身の身元をSMTPサーバに対し示します。パラメータにはクライアントのホスト名を

注3) ISPや、第三者組織が送信者の信頼度をスコアリングした指標です。正常な送信を続けていると高くなり、低い場合は受信を拒否されるリスクが上昇します。
注4) 本記事で実際に使用するコマンドだけを記載しています。
注5) SendGrid以外のSMTPサーバを利用する場合は、例とは多少異なる可能性があります。また、使用する送信元および宛先メールアドレスは、すべてご自身のメールアドレスを使用してください。 URL https://sendgrid.kke.co.jp/
注6) 25、465、2525のポートも使用できますが、SendGridの推奨する587を使用します。



入力します。レスポンスとしてサーバのホスト名およびそのサーバが対応している拡張機能の情報が返されます。

```
EHLO mail.example.com Enter
250-smtp.sendgrid.net ←サーバのホスト名
250-8BITMIME ←以下、対応している拡張機能
250-PIPELINING
250-SIZE 31457280
250-STARTTLS
250-AUTH PLAIN LOGIN
250 AUTH=PLAIN LOGIN
```

Step3「ログイン」

SendGridではSMTP認証が必須になっているので、ログイン処理を実施します。今回はAUTH LOGIN認証を使用します。「AUTH LOGIN」コマンドを送り、その後ユーザ名とパスワードをBase64でエンコード[注7]して送ります。

```
AUTH LOGIN Enter
334 VXNlcm5hbWU6 ←SMTPサーバがユーザ名を要求している
dXNlcm5hbWU= Enter ←ユーザ名をBase64エンコードした値
334 UGFzc3dvcmQ6 ←SMTPサーバがパスワードを要求している
cGFzc3dvcmQ= Enter ←パスワードをBase64エンコードした値
235 Authentication successful
```

Step4「送信元メールアドレスの指定」

続いて、MAILコマンドを利用して、送信元アドレスを設定します。

```
MAIL FROM:<sender@example.com> Enter
250 Sender address accepted
```

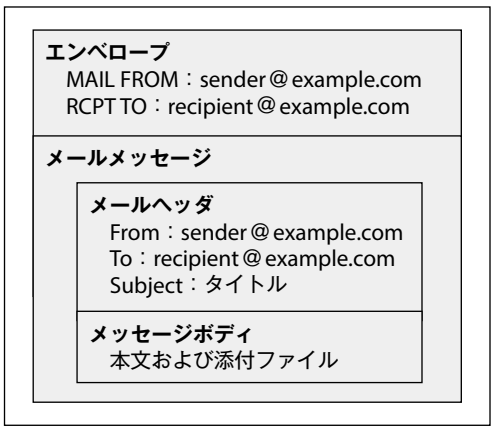
Step5「宛先メールアドレスの指定」

さらに、RCPTコマンドで宛先アドレスを設定します。

```
RCPT TO:<recipient@example.com> Enter
250 Recipient address accepted
```

ここまででいわゆるメールのエンベロープと呼ばれる部分の設定が完了です。ここから、メールメッセージ(メールヘッダおよびメッセージボディ)の部分を設定していきます。メールはエンベロープとメールメッセージで構成され、エンベロープは配送に必要な情報が記載されている封筒のようなもの、メールメッセージはその封筒に入っている便箋(びんせん)のようなものだと考えてください(図2)。

▼図2 メールの構成



Step6「メッセージの入力」

メッセージはメールヘッダとメッセージボディで構成され、DATAコマンドから「.」+「改行(<CRLF>)」までがメッセージとして扱われます。

```
DATA Enter
354 Continue
To:recipient@example.com Enter
From:sender@example.com Enter
Subject:Test Email Enter
Enter ←空行でヘッダとボディを分ける
Sample message body
. Enter ←.<CRLF>でボディの終端であることを示す
250 Ok: queued as _c210cC13JpZC5uZXQ=
↑正常に受付完了
```

Step7「切断」

以上でメール送信リクエストが完了したので、QUITコマンドでSMTP接続を切断します。

```
QUIT Enter
221 See you later
Connection closed by foreign host.
```

Step8「確認」

宛先に指定したメールアドレスに実際にメー

注7) こちらのサイトを利用するとBase64エンコード/デコードともに簡単に行うことができます。
URL https://www.base64encode.org/

▼図3　実際に届いていたメール



ルが届いているか確認します。筆者の場合はGmailのアドレスを宛先にし、**図3**のように届いていました。



送信ドメイン認証

なりすましが容易なメールと、それを防ぐための取り組み

前述の手順でわかるとおりSMTPではエンベロープ、ヘッダのToおよびFromを自由に指定することが可能なため、極端な話、エンベロープのTo以外すべてニセの情報を設定しても送信可能で、また、もともと送信者を認証するしくみもありませんでした。そのシンプルさを悪用し、迷惑メールや標的型攻撃メールといった悪意をもった用途での利用が後を絶たず、認証やレピュテーションによって正当な送信者であるかを判断するしくみができてきました。

メールに関する業務の経験がある方はSPF（Sender Policy Framework）やDKIM（DomainKeys Identified Mail）といったキーワードを聞いたことがあるのではないでしょうか。これらは送信ドメイン認証の手法で、送信者のドメインが正しいかを判定するのに利用されます。

送信ドメイン認証のしくみ

SPFもDKIMも送信者が自身のドメインのDNSに事前に必要な情報を登録しておき、受信者がメールを受け取る前にそのレコードを確認することで、送信者の認証を行います（**図4**）。

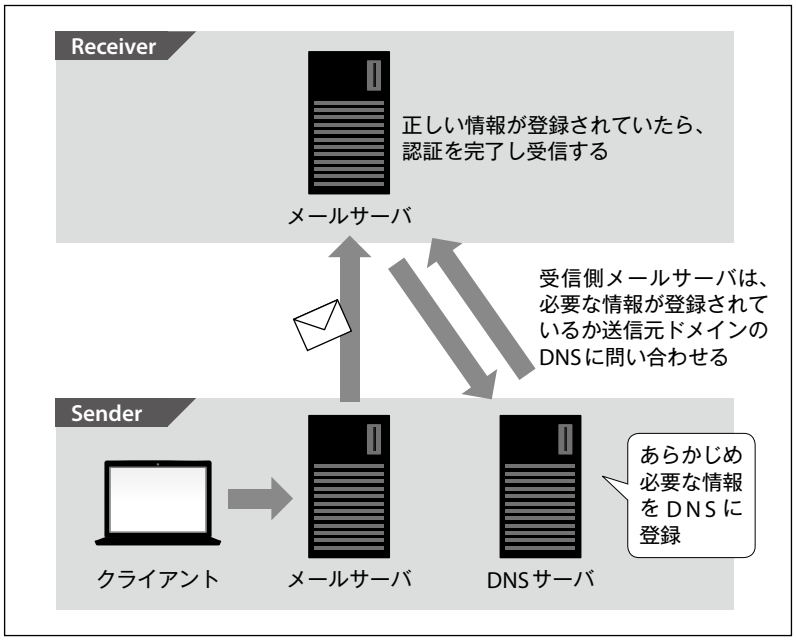
SPFの場合は、送信者が事前に使用するメールサーバのIPアドレスをすべてDNSサーバに列挙します。受信者は実際の接続元IPアドレスと比較することで正当な送信者かどうかを判断します。

DKIMの場合は、DNSサーバに公開鍵を設置しておきます。DKIMは公開鍵方式を利用した認証手段で、送信者は秘密鍵を利用して生成した署名をメールヘッダに付与して送信します。受信者は公開鍵を利用してメールヘッダの署名が正常か検証します。これによりメールが改ざんされていないことを確認することができます。

nslookupコマンドで、送信ドメイン用レコードを確認してみる

先ほどとは逆に、今度はメールを受け付ける

▼図4　送信ドメイン認証のしくみ





立場である宛先サーバがどのような認証処理を行っているか、nslookupコマンドで実際に体験してみましょう。

SPF

SPFに関する情報はエンベロープのFromのドメインに対し、TXTレコードで規定の書式に従って登録されています。たとえばSendGridドメインのTXTレコードを確認してみましょう。

```
$ nslookup -type=TXT sendgrid.net Enter
text = "v=spf1 ip4:167.89.0.0/17 ip4:208
.117.48.0/20 ip4:50.31.32.0/19 ip4:198.
37.144.0/20 ip4:198.21.0.0/21 ip4:192.254
.112.0/20 ip4:168.245.0.0/17 ~all"
```

このように、IPアドレスが列挙されたレコードが返却されます[注8]。宛先サーバでは送信元ドメインのTXTレコードと、実際の送信元IPアドレスが合致するか確認します。

この認証処理は、宛先サーバで自動的に行われるため通常気にする必要はありませんが、認証結果自体はメールのソースから確認ができます。Gmailで受信した場合は次のようなヘッダが記載されます(次は正常にpassした場合の例)。

```
Received-SPF: pass (google.com: domain of
sender@example.com designates 192.0.2.0
as permitted sender) client-ip=192.0.2.0;
```

DKIM

DKIMの認証時に確認すべきTXTレコードの情報は、メールヘッダに付与される署名内に記載されています。たとえば、SendGridから送信された場合、次のような署名が付与されます。

```
DKIM-Signature: v=1; a=rsa-sha1; c=relaxed
/relaxed; d=sendgrid.info; h=from:to:
subject; s=smtpapi; bh=36gYllQgpA7ZWa26ry
LrOZA0+io=; b=gPvxfw3p3DVuUCBD+30D7I4X7St
/a fFh1qQLSoRPniqhYBM3KE 【以下略】
```

DKIM署名には、「d=」「s=」といったさまざまなタグが使用されていますが、

```
<sタグの値>._domainkey.<dタグの値>
```

のTXTレコードに公開鍵を設置するよう定められています。上記の例では、smtpapi._domainkey.sendgrid.infoを確認すれば良いことになります。実際にやってみましょう。

```
$ nslookup -type=TXT smtpapi._domainkey.
sendgrid.info Enter
smtpapi._domainkey.sendgrid.info  text =
"k=rsa; t=s¥; p=MIGfMA0GCSqGSIb3DQEBAQUA
A4G 【中略】 d6309d8z1XkUBwIDAQAB"
```

宛先サーバではこの情報をもとに署名が正しいか確認をします。



まとめ「プリミティブな道具で知識と技術を磨く」

本記事では身近なメールを題材として取り上げ、telnetを利用したSMTP通信、nslookupを利用した送信ドメイン認証の体験をしました。おそらく読者の方の中で両方を実際に試したことがある方は少ないのではないでしょうか。

SMTPの通信はメールクライアントや各種ライブラリによってラッピングされているため通常触ることはありませんし、送信ドメイン認証も宛先サーバで自動的に行われるため普通は確認することはありません。

便利なアプリケーションやライブラリが世の中にはたくさん用意されているため、今回取り上げたようなプリミティブな技術を触る機会はかなり少なくなってきています。しかし、たとえばメールが送信できない、届かない、といったトラブルが発生したときには、これらを知っているかどうかが、解決までの時間に大きく影響してきます。もちろん便利なツールを使用するのは良いのですが、たまにはこういった技術に触れてみてはいかがでしょうか。SD

注8) レコードの詳しい見方は次のURLを参照ください。
URL https://sendgrid.kke.co.jp/blog/?p=3509

書いて覚える Swift 入門

第30回 SD読者ならわかるブロックチェーン入門



Author 小飼弾（こがい だん）　twitter @dankogai

前号の予告に変えて

前回の記事ではSwiftのSequenceプロトコルを取り上げたうえで、次回はそのSequenceの中でもランダムアクセス可能なCollectionを取り上げる予定でしたが、もう1回ある特殊なSequenceを取り上げることにします。それも、今後のITの世界を一変させる可能性を秘めた、とあるSequenceです。それはSwiftのコードにすれば、

```
protocol Block {
    associatedtype Signature
    associatedtype Payload
    func isValid(prev:Self)->Bool
}

enum BlockError: Error {
    case Invalid
}

protocol BlockChain : Sequence {
    associatedtype Element:Block
    var last:Element? { get }
    mutating func append(_ block:Element)
throws

}
```

と表現されるSequence、すなわち**ブロックチェーン**という名のSequenceです。

ブログ[注1]でも発表したとおり、このたび私、小飼弾は株式会社VALUのリードエンジニアに就任いたしました(**図1**)。同社のサービスであるVALUはVAという個人の指標をビットコイン($BTC)で売買する市場を備えたSNSで、その売買手数料が同社の収益源なのですが、それは売上が100%ビットコインだということ。ビットコインあってのVALUなのです。

そしてそのビットコインは、ブロックチェーン(blockchain)という技術で成り立っています。ビットコインはブロックチェーンの最初にして

▼図1　筆者のVALU



注1）URL http://blog.livedoor.jp/dankogai/archives/52031757.html

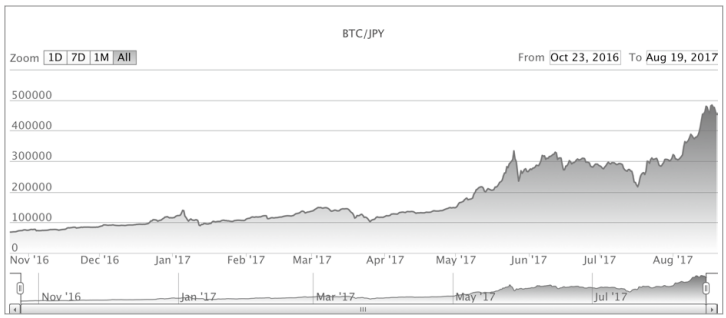
現時点における最重要のアプリケーションではありますが、ブロックチェーンでできることの氷山の一角に過ぎません。

断言します。ブロックチェーンはWWW以来、最重要のIT(Information Technology)である、と。

本誌の読者であれば、今年中にHTTP(S)やHTMLなみにごく当然のようにわきまえておかねばならない常識になる、と。

と言っても慌てる必要はありません。ブロックチェーンが今後不可欠な基盤技術であることと受け入れられたのはごく最近なのですし、理解と習得に何年もかかるほど難解なものではないのですから。**図2**のチャートをご覧ください。これは日本国内最大級の交換所、bitFlyer社の2017年08月19日における、ビットコインと日本円の交換レートなのですが、注目していただきたいのは急騰している右上ではなく左下。Nov '16とあります。同社のチャートでさえこれしか歴史がないのです。今から学習しても十分間に合います。

▼図2　https://bitflyer.jp/ja/bitcoin-chart



Welcome to the blockchain!

ブロックチェーンとはいったい何か？　**刻々と追記されるデータをP2Pネットワークで安全確実に同期する技術**。これだけです。どうやってそれを実現しているかを知る前に、まずは「すでに記録が完了したデータをP2Pネットワークで安全確実に同期する」方法をおさらいしておきましょう。

「すでに記録が完了したデータ」と言えば、ファイル。P2Pと聞いて本誌の読者の皆さんが思い出すのは、WinnyやBitTorrentといった、ファイルを取得するためのP2Pではないでしょうか。誰からもらったのかもわからないファイルが改ざんされていないことを、我々はどう確かめたら良いのでしょう？　そのファイルに固有の特徴(signature)を見ていますよね。

たとえばOSのインストーライメージであれば、md5やSHA-1やSHA-256といったチェックサム(checksum)が記載されたファイルが必ず配布ディレクトリに配置されています。なので ubuntu-16.04.3-desktop-amd64.iso[注2] のSHA-256ダイジェストが1384ac8f2c2a6479ba2a9cbe90a585618834560c477a699a4a7ebe7b5345ddc1であることを確かめれば、正体不明の人からそれをもらっても間違いなく正しいファイルであることを確認できるわけです。

ファイルであれば、ファイルそのものは信頼できない経路でもらっても、signatureさえ信頼できる経路でもらえばOKですが、それを「終わりなきデータ」で実現するにはどうしたら良いのでしょうか？

適当にデータがたまったら、それをファイルにして、そのファイルの前のファイルのsignatureをわかるようにしておけばいい。これがブロックチェーンの基本的な考えで、基本はそれだけです。この「すでに記録が完了した」

注2）URL http://ftp.riken.jp/Linux/ubuntu-releases/16.04/ubuntu-16.04.3-desktop-amd64.iso
URL http://ftp.riken.jp/Linux/ubuntu-releases/16.04/SHA256SUMS

データを、「ファイル」ではなく「ブロック」と呼称しているだけです。次は前のブロックのhashvalueを署名とする最も簡単なブロックチェーンの実装です。

```
struct SimpleBlock<D:Hashable> : Block {
    typealias Signature = Int
    typealias Payload = D
    let prevSign:Signature
    let payload:D
    func isValid(prev:SimpleBlock<D>)->Bool {
        return prev.payload.hashValue ==
 self.prevSign
    }
}

struct SimpleBlockChain<D:Hashable> :
BlockChain {
    typealias Element = SimpleBlock<D>
    var blocks:[Element]
    var last:Element? {
        return blocks.last
    }
    mutating func append(_ block:Element)
throws {
        if blocks.isEmpty || block.
isValid(prev: last!) {
            blocks.append(block)
        } else {
            throw BlockError.Invalid
        }
    }
    func makeIterator() -> AnyIterator<Element> {
        var count = 0
        return AnyIterator {
            if count == self.blocks.count {
                return nil
            } else {
                defer { count += 1 }
                return self.blocks[count]
            }
        }
    }
}
```

次の処理結果を見てのとおり、直前のブロックのpayloadのhashValueが一致しないとappendに失敗します。

```
extension SimpleBlockChain {
    init() { blocks = [] }
}

var sbc = SimpleBlockChain<String>()

do {
    try sbc.append( // 最初のブロック
      SimpleBlock(prevSign: 0, payload: "Zero")
    )
    try sbc.append( // 次の不正なブロック
      SimpleBlock(prevSign: 0, payload: "1")
    )
} catch {
    print(error)
}
try sbc.append(
    SimpleBlock(  // 次の正しいブロック
       prevSign: 4799450038612711877,
       payload: "one"
    )
  )
for b in sbc {
    print(b.payload)
}
```

しかしこれだと、あるデータの続きを作るだけなら誰でもできてしまうことになります。前にあるデータのsignatureは誰にでも作れてしまうのですから。つまりデータはフォーク（fork）する可能性があるのです。実際ビットコインも今年8月1日にフォークしたのですが、ブロックチェーンが保証するのはその前までのデータであって、その次以降のデータはその限りではないということは、大事なことなので忘れないようにしておいてください。

しかしそれではフォークだらけになって収拾がつかなくなるのは目に見えています。それを防ぐにはどうしたら良いでしょう？　ここで紙の世界を見てみましょう。複数のページからなる契約書があるとします。この契約書がひとつながりの契約書であることをどう保証しているか？　全ページに同じ人が署名するのです。実際には各ページにはイニシャル程度の略式の署名をし、最後のページに本署名するのが現在確

立されたスタイルですが、要は誰でも勝手に署名できない形で署名してしまえば、紙であろうがブロックであろうが、それがひとつながりのデータであることは保証できるわけです。

中央集権から分散へ

本誌の読者であれば電子署名がすでに枯れた技術であることはご存じかと思います。iOSやAndroidのアプリはすべてストアによって電子署名されていて、アプリを構成するデータがそのストアで配布されたことを保証しています。だからこそアプリストアが成り立つわけですが、だからこそアプリストアは中央集権なのです。アプリに署名できるのはストア主だけなのですから。

これを分散するにはどうしたら良いでしょう？複数の人が署名できるようにするだけならさほど難儀でありません。署名する権限がある人のリストを用意しておき、誰が署名したかをブロックごとに検証していけば良いだけです。しかしこれでも署名者登録が中央集権的にならざるをえません。世界最重要の分散型データベースであるDNSも、ルートサーバはInternicという組織がトップダウンに配布しています。

分散型ネットワークであるインターネットも、このように分散不可能な要素を抱えているのです。これを真の意味で分散可能にできないか？　ブロックチェーンであればその根幹である署名プロセスを分散できないか？

Bitcoinのしくみ

そこに颯爽と登場したのが、Satoshi Nakamoto[注3]を名乗る1人の天才でした。彼は特権なしで誰でも署名できる一貫性のある方法を論文で提示[注4]したのです。

その方法は実にシンプルなものでした。

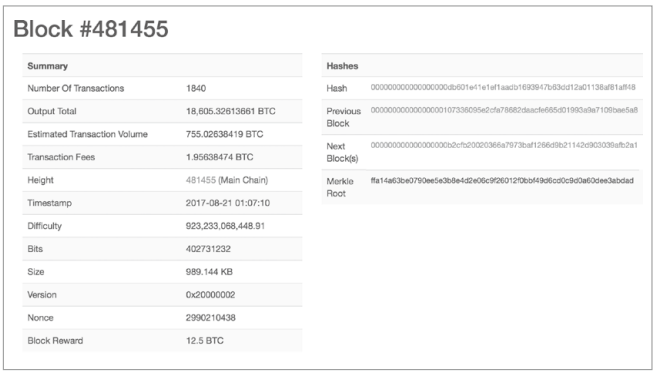
①最初に署名した人が正しい署名ということにする
②ただし署名するにはそれなりの労力をかけてもらう
③その労力に対し代価を支払う（ビットコインで）
④署名できなかった人はその署名を検証する

あとは、「それなりの労力がかかる」方法さえあればいい。Bitcoinではその方法は次のとおりです。

①署名元のデータに、32bitのnonceというデータを足す
②それに電子署名する
③その署名結果、つまり2進法の数値が一定以下であれば正しい署名として認める。それ以外は不正な署名とする

図3はblockchain.infoによる481455番目[注5]のbitcoin blockの内訳ですが、右側にHashesという値があります。16進数表示で64桁、つまり256bitの数値ですがこれが署名です。いずれも頭に0がいくつも並んでます。この0が

▼図3　481455番目のbitcoin blockの内訳



注3）URL https://en.wikipedia.org/wiki/Satoshi_Nakamoto
注4）URL https://bitcoin.org/bitcoin.pdf
注5）URL https://blockchain.info/block/00000000000000000db601e41e1ef1aadb1693947b63dd12a01138af81aff48

一定個数以下のものだけ、正しい署名と認めることにするのです。

この署名を得るために、署名者はnonceを変えては署名して、不正ならnonceを変えてまた署名……という作業を延々と繰り返します。この作業のことを業界では「採掘(mining)」と呼んでいますが、そうして正しく署名したブロックをP2Pネットワークに流し、ほかのネットワークノードに検証してもらって初めて、そのブロックは正当なブロックとして承認されます。

正しい署名が見つかる確率を難易度(difficulty)と言いますが、ビットコインでは10分に1ブロック増える程度に都度調整されるようになっています。「送金に10分」というのはこれが理由です。送金記録がブロックに掲載され、それがチェーンにつながれたとき、その取引は正当なものとなるのです。

そう。送金記録。48万個を超えるブロックの連鎖からなり、今も10分に1つずつ増えていく、過去と未来すべての入出金の歴史。それこそがビットコインの真の姿。その意味ではそれを「台帳」(ledger)ではなく「コイン」という言葉を選んだSatoshi Nakamotoは罪作りではあります。しかし「コイン」という言葉の沿革を見ると、ビットコインという命名はなかなか渋い。中央銀行以上に信用があり得るものといえば金[注6]で、金でできた貨幣をコインと呼ぶのですから。

押さえておくべき3ポイント

ここでブロックチェーンに話を戻すと、見てのとおりブロックチェーンとは刻一刻と付け加えられるデータを全員で共有するための技術に過ぎず、どんなデータでも流せることは明らかです。分散署名も署名者を固定すれば不要で、たとえば日銀が電子署名する「日銀チェーン」だってあり得ますし、クローズドな環境ならば電子署名さえ不要で、さらにはログの共有などに用いても何の問題もありません[注7]。

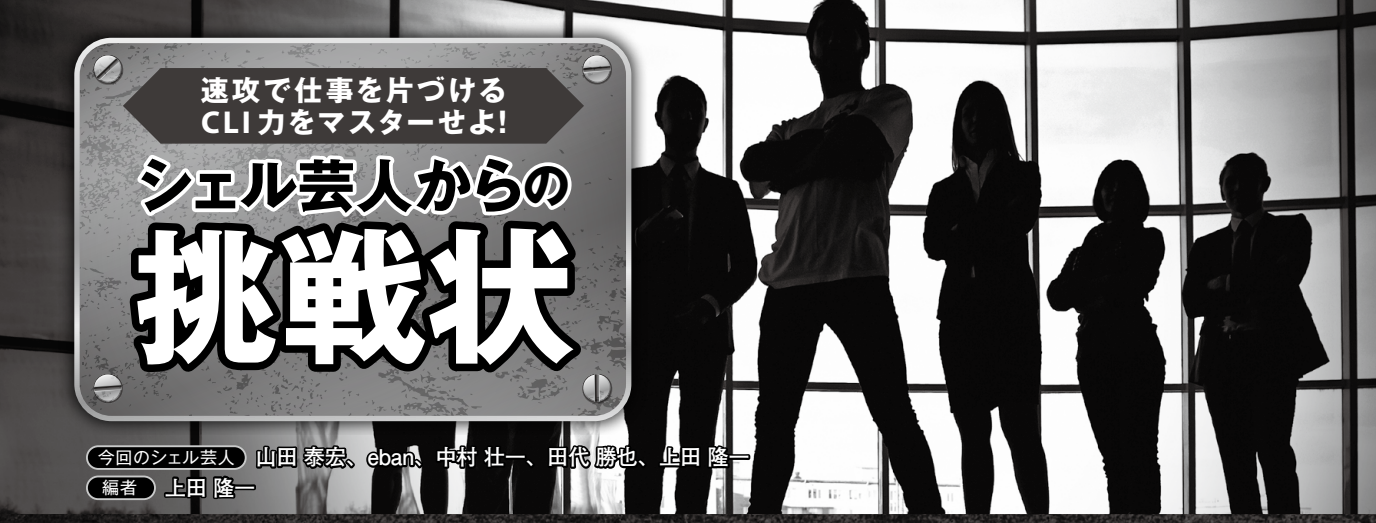
とはいえブロックチェーンという言葉がバズるところまで来たのは、何と言っても分散署名の方法をNakamotoがBitcoinとともに提示し、実際にそれを用いた取引が根付いてから。Nakamoto論文の公開は2008年。そこから10年足らずで「ただの思いつき」はビットコインだけでも約7兆円、仮想通貨全体では14兆円もの経済活動をカバーするにいたっています[注8]。IPOの代わりの資金調達手段としてICO(Initial Coin Offering)というものも登場しましたし、ブロックチェーンの世界の片隅で暮らし始めた筆者にもブロックチェーンの適応と放散の速度には眩暈(めまい)を禁じえません。

そろそろ紙幅も尽きるのでまとめます。

1. ブロックチェーンとは刻々と追記されるデータをP2Pネットワークで安全確実に同期する技術
2. チェーンにブロックを追加するには、追加前のブロックを署名する
3. Satoshi Nakamotoが提示したのは、誰がブロックを用いて署名してもチェーンが切れない方法。ビットコインはそれを使って入出金記録を記録し続けるためのプロトコル(protocol)

これだけ押さえておけば、ブロックチェーンにはついていけます。読者のあなたの学習チェーンにぜひこの知識をつなげてみてください。SD

注6) 日本銀行の貸借対照表の資産の部の一番上にあるのはなんでしょう？
注7) こういったブロックチェーンはプライベートチェーンと呼ぶことが多いようです。
注8) URL https://coinmarketcap.com



速攻で仕事を片づける
CLI力をマスターせよ!

シェル芸人からの挑戦状

今回のシェル芸人 山田 泰宏、eban、中村 壮一、田代 勝也、上田 隆一
編者 上田 隆一

第2回 暦や日付の計算

端末上で日付計算をする意味

今回は暦や日付を扱う問題を解いてみます。日付や時間の計算はバグや事故、プログラマの悲鳴と罵倒(ばとう)を日々生産しています。このややこしさはおそらく本質的なもので、ツールを使ったからと言って簡単になるものではありません。

しかし、端末上で日付の計算ができると、わずかながらもストレスを減らせるかもしれません。プログラムを書いた前後の検証にも便利です。そこで、ややこしい問題を解くことで細かいテクニックを身につけていただこうというのが今回の趣旨です。

問題の検証に使ったOSはUbuntu Server 16.04 LTSです。Ubuntuですので、基本的なコマンドはGNU core utilitiesの使用を前提にしています[注1]。シェルは前回同様bash 4.3です。

注1) そのほかのコマンドについては、さまざまなものを試すのでその都度インストールします。解説においてインストールが必要なことを説明します。ただ、aptで簡単にインストールできるものは手順を省略することがあります。

問題1 休日の突き合わせ

初級 ★

出題、解答、解説：山田

少し前に話題になりましたが、内閣府がWebサイトで日本の祝日の一覧のCSVファイルを配布しています。「http://www8.cao.go.jp/chosei/shukujitsu/syukujitsu.csv」からダウンロードできます。

このsyukujitsu.csv(文字コードはShift_JIS)には、リスト1-1のような文字列が入っています。このファイルを利用して、リスト1-2のような2016-01-01から2018-12-31までの期間の日付一覧を作ってください。その際、祝日でない日には㋀を補ってください。出力の文字コードはUTF-8とします(解答例のファイルは2017年7月23日に取得されたものです)。

▼リスト1-1 syukujitsu.csv

```
国民の祝日月日,国民の祝日名称
2016-01-01,元日
2016-01-11,成人の日
2016-02-11,建国記念の日
2016-03-20,春分の日
(..略..)
2018-11-23,勤労感謝の日
2018-12-23,天皇誕生日
```

▼リスト1-2 2016-01-01から2018-12-31までの日付一覧

```
2016-01-01,元日
2016-01-02,㋀
(..略..)
2018-12-22,㋀
2018-12-23,天皇誕生日
(..略..)
2018-12-31,㋀
```

解答

図1-1に解答例を示します。順を追って説明します。まずはechoコマンドで2016年から2018年までの日付の一覧を列挙しています。最初に{ }を使った記述が与えられています。こ

▼図1-1　解答

```
$ echo 201{6..8}{01..12}{01..31} | tr ' ' '\n' | date -f - +%F 2>/dev/null | sed 's/$/,@/' |
cat - <(iconv -f SJIS -t UTF-8 syukujitsu.csv | tr -d '\r' | awk 'NR>1') | sort -r | uniq -w10 | tac
```

れはbashのブレース展開と呼ばれる記法です(**図1-2**)。連番を出力するのによく使います。

ブレース展開の結果は空白で区切られるので、解答例ではtrコマンドで1行に1つに折り返し、**図1-3**のような出力を作っています。

しかしこの方法だと、存在しない日付、たとえば2月31日などが出現してしまいます。これを除去するために、解答例では`date -f - +%F`というコマンドに通しています。この結果、**図1-4**のように存在する日付は標準出力から出て後続のsedで末尾に`,@`が付加され、それ以外はエラーメッセージとして標準エラー出力に振り分けられます。解答例では標準エラー出力を捨てる`2>/dev/null`という記述をして、dateコマンドのエラーメッセージを消しています。GNUのdateは`-f`でファイルから読み込み、`-f -`で標準入力から日付を1つずつ読み込みます。

▼図1-2　ブレース展開の例

```
$ echo {01..03}
01 02 03
↓組み合わせも出力可能(数学でいう直積)
$ echo {01..03}{10..13}
0110 0111 0112 0113 0210 0211 0212 0213
0310 0311 0312 0313
```

▼図1-3　日付を出力(2月31日など不正な日付も含む)

```
$ echo 201{6..8}{01..12}{01..31} | tr ' ' '\n'
20160101
20160102
(..略..)
20181231
```

▼図1-4　存在しない日付は標準エラー出力に出る

```
$ (図1-3のコマンド) | date -f - +%F | sed 's/$/,@/'
(..略..)
2016-02-28,@   ┐
2016-02-29,@   ┘ 標準出力から
date: invalid date '20160230'   ┐
date: invalid date '20160231'   ┘ 標準エラー出力から
2016-03-01,@   ←標準出力から
(..略..)
```

dateは8桁の数字を読み込むと日付(西暦4桁、月2桁、日2桁)として解釈してくれます。さらに解答例のdateには`+%F`という引数を与えていますが、これで出力が**図1-4**のように「*西暦4桁-月2桁-日2桁*」の形式になります。

その次の`cat - <()`では、まず標準入力からきた字がcatコマンドでそのまま出力され、その後に`<()`の処理の中身が出てきます(**図1-5**)。`<()`はbashのプロセス置換という機能です。catは`<()`をファイルとして扱います。そのファイルの中身は`<()`内の処理の結果になります。

プロセス置換の中の処理は、まずiconvコマンドでShift_JISで記述された「syukujitsu.csv」をUTF-8に変換しています。次のtrでWindowsの改行(CRとLFの2種類のコードで表現された改行)からCRを削除し、最後に`awk 'NR>1'`で2行目以降のみを出力(1行目のヘッダを除去)しています。awkでNRは行番号を意味します。

次にこれを後続の`sort -r`(rは「逆順」)で並び替えすると、祝日のみが2回ずつ連続する**図1-6**の出力が得られます。これで、祝日のある

▼図1-5　日付の一覧にsyukujitsu.csvを結合

```
$ (図1-3のコマンド) |
date -f - +%F 2>/dev/null | sed 's/$/,@/' |
cat - <(iconv -f SJIS -t UTF-8 syukujitsu.csv |
tr -d '\r' | awk 'NR>1')
2016-01-01,@
2016-01-02,@
(..略..)
2016-01-01,元日
2016-01-11,成人の日
(..略..)
```

▼図1-6　祝日では日付が2つずつ出力される

```
$ (図1-5のコマンド) | sort -r
(..略..)
2018-12-24,@
2018-12-23,天皇誕生日
2018-12-23,@
2018-12-22,@
(..略..)
```

日はレコードが2つ、祝日名のあるほうが先に出力されます。重複はuniq -w10で除去します。uniqコマンドは通常、同一内容の行を1行にまとめますが、-w10を与えると文頭から10文字(日付)のみを比較し、先にあるデータだけ出力します。そして最後にtacで逆順だったものを戻すと、**リスト1-2**のような出力が得られます。

別解

細かい説明は割愛しますが、**図1-7**はjoinコマンドとそのオプションをうまく活用した、たいへんスマートな解答例です。

▼図1-7　別解(eban)

```
$ seq -f '2016-1-1 %g days' 0 $((365*3)) | date -f- +%F | join -t, -a1 -e@ -o1.1,2.2 - <(nkf -wLu ↵
syukujitsu.csv|sed 1d)
```

問題2　プレミアムフライデー

初級 ★

出題、解答、解説：中村

2017年2月24日より、プレミアムフライデー(以下、PF)が施行されました。月末の金曜日がPFです。2017年2月24日から2017年末までの、PFの一覧を表示してください。

解答

最初におおまかな日付の一覧を作成しておき、少しずつ対象の日付を絞り込んでいく戦略で解いていきます。まずは、2017年2月24日からだいたい1年間の日付を列挙します(**図2-1**)。

dateの-d STRINGオプションは、STRINGに指定された日付時刻を表示するオプションです。ここにNdayと指定すると、その前に記述した日付からN日後の日付を出力することができます。たとえば-d '2017-02-24 3day'は2017-02-27になります。

```
$ date -d '2017-02-24 3day'
2017年  2月 27日 月曜日 00:00:00 DST
```

図2-1では、seqコマンドとxargsコマンドによって@dayの部分が0day、1day、2day～365dayと変化していくため、366日分の日付が出力されます。

ここからデータをふるいにかけていきましょう。まずは「2017年」と「金曜日」だけを抽出します(**図2-2**)。その次は月末の金曜日を抽出します。月末ということは、月が変わる直前の行ですので、月までの内容(先頭から7文字)で重複カットしてあげれば抽出できそうです(**図2-3**)。

▼図2-2　「2017年」かつ「金曜日」の日付を抽出

```
$ (図2-1のコマンド) | grep ^2017 | grep '金'
2017-02-24 金
2017-03-03 金
2017-03-10 金
(..略..)
```

▼図2-1　2017年2月24日から366日間のカレンダーデータ(うるう年でも大丈夫)

```
$ seq 0 365 | xargs -I@ date '+%F %a' -d '2017-02-24 @day'
2017-02-24 金
2017-02-25 土
2017-02-26 日
(..略..)
```

▼図2-3　問題1と同じくuniq -w7を利用して抽出

```
$ (図2-2のコマンド) | tac | uniq -w7 | tac
2017-02-24 金
2017-03-31 金
(..略..)
2017-12-29 金
```

▼図2-4　解答

```
$ seq 0 365 | xargs -I@ date '+%F %a' -d '2017-02-24 @day' | grep ^2017 | grep '金' | tac | uniq -w7 | tac
```

これは1問目と同じ展開です。ということで、最終的な解答例は**図2-4**のようになりました。

別解

日付を便利に扱えるdateutilsというコマンド群があります。このdateutilsのdroundを使ったスマートな別解を示します(**図2-5**)。dateutilsはsudo apt install dateutilsでインストールできます。

▼図2-5　dateutils.droundを利用した別解(田代)

```
$ printf '2017-%02d-01\n' {2..12} | ↵
dateutils.dround -- +31d -Fri
2017-02-24
2017-03-31
(..略..)
```

問題3

UNIX時刻の限界

出題：田代　解答、解説：上田

中級 ★★

お使いのシステムのdateコマンドで、利用できるUNIX時刻の最大値をワンライナーで求めてください。UNIX時刻は、世界標準時の1970年1月1日からの秒数です。

解答

これは時刻の計算というより二分探索(バイナリサーチ)をワンライナーで実装する問題になります。答えを先に示します(**図3-1**)[注2]。

この出力の下から2行目の67768036191644399が筆者(上田)の使っているシステムでのUNIX時刻の最大値で、最下行の2147485547年 12月 31日 水曜日 23:59:59が年月日表示での最大値になります。

まず、**図3-1**の1、2行目で探索範囲を設定しています。変数fに0、変数tに大きな整数を代入しています。あとの行でfとtの間に求めたい時刻が必ず入るようにしながらfとtの間を縮めていきます。

2行目がややこしいので解説します。まず、bcは計算のためのコマンドです。たとえば2の100乗を計算したければ、次のように、

```
$ echo 2^100 | bc
1267650600228229401496703205376
```

と標準入力に計算式を入れて使います。**図3-1**の2行目では、echoを実行してパイプで式を渡す代わりにbashの<<<を使ってbcに式を渡しています。bcを囲っている$()は標準出力から出てきた数字をbashの文字列に変換するためのものです。これでtにbcの出力が代入されます。

3行目の解説を後回しにして4行目に飛びます。ここではbcを使ってfとtの真ん中の値を求めています。5行目でこの数字を出力したあと、6行目でdateに渡しています。date -d @*UNIX時刻*で、指定したUNIX時刻を年月日時分秒表記に変換していますが、ここは「指定されたUNIX時刻が変換可能かどうか」を判断するためにdateを使っています。**図3-2**のように、UNIX時刻の桁が大きいとコマンドはエラーを返します。

図3-1の6行目の後半は、dateの結果で条件分岐しており、dateが成功すれば変数mをfに、失敗すればmをtに代入しています。これで、dateがfで成功、tで失敗する状態が保たれつつ、f

▼図3-1　解答

```
$ f=0 ; ↵
t=$(bc <<< 2^100) ; ↵
while [ $(bc <<< $t-$f) != 1 ] ; do ↵
m=$(bc <<< "($f+$t)/2"); ↵
echo $m; ↵
date -d @$m && f=$m || t=$m ; ↵
done
(..略..)
67768036191644398
2147485547年 12月 31日 水曜日 23:59:58 JST
67768036191644399
2147485547年 12月 31日 水曜日 23:59:59 JST
```

注2) 字数を切り詰めるために通常クオートするところをしていません。

とtの範囲が狭くなっていきます。

最後に3行目ですが、これはfとtの差が1になったかを[コマンド(テストコマンド)で確認し、1になったらループを抜けるという記述です。[コマンドで数字を比較するときは-eqなどのオプションを使いますが、桁数が大きいとエラーが起きるので文字列として比較しています。

▼図3-2　指定したUNIX時刻が利用可能かどうかをチェックする

```
$ date -d @123
1970年  1月  1日 木曜日 09:02:03 JST
$ echo $?   ←終了ステータスを確認
0           ←OKの場合
$ date -d @12345678901234567 8
date: 時間 `12345678901234567 8' が範囲外です
$ echo $?
1           ←NGの場合
```

問題4　リスケジュール

上級 ★★★

出題、解答、解説：田代

2017年9月18日に打ち合わせをする予定がありました。しかし、都合で次週の火曜日に変更になりました。さらに次週の金曜日に変更になりました。さらに前日に変更になりました。打ち合わせは何月何日になったでしょうか(ワンライナーでなくてもかまいません)。

解答

まずは確認ですが、この問題は「**2017年9月18日(月) → 次週の火曜 → 次週の金曜 → 前日**」のように日を追っていけば解けます。

dateコマンドではnext weekやyesterday、next Fridayなど相対的にずらした日付を指定できます。info '(coreutils) date invocation'を実行して表示されるマニュアルのDate input formatsの節や、GNU公式Webページ[注3]に詳しい解説があります。この指定方法で、後ろにつなげていくと、日を追っていけます。

```
$ date
2017年  8月  2日 水曜日 01:37:38 JST
$ date -d 'next week'
2017年  8月  9日 水曜日 01:37:50 JST
$ date -d 'next week next Friday'
2017年  8月 11日 金曜日 00:00:00 JST
$ date -d 'next week next Friday yesterday'
2017年  8月 10日 木曜日 00:00:00 JST
```

次は指定した日付から相対的にずらす例です。

```
$ date -d '2017-09-18 yesterday'
2017年  9月 17日 日曜日 00:00:00 JST
```

注3) https://www.gnu.org/software/coreutils/manual/html_node/Relative-items-in-date-strings.html#Relative-items-in-date-strings

```
$ date -d '2017-09-18 next week'
2017年  9月 25日 月曜日 00:00:00 JST
```

ただ、残念ながらnext Tuesdayのような曜日の指定は機能しないようです。

```
$ date -d '2017-09-18 next Tuesday'
2017年  9月 18日 月曜日 00:00:00 JST
```

そこで、次のようなコマンドnextwdを作ってみましょう。

```
↓次週の火曜日
$ echo 2017-09-18 | nextwd Tue
2017-09-26
```

bashの関数を作ってしまえば、シェル上でコマンド(のように動くもの)が作れます。基準日を標準入力から受け取り、曜日を引数で指定する関数nextwdを**図4-3**のように記述します。

この関数は先月号第1回の問題「問題1」を参考にして作りました。標準入力から日付(YYYY-MM-DD形式)を1つ読み込んで変数dに入れ、引数で曜日を受け付けます。たとえば標準入力に2017-09-18、引数にTueを入れると、$()の中身は**図4-4**のようになります。そして、一番外側のdateの出力は**図4-5**のようになります。

以上から、関数の定義込みで1行で書いた**図**

4-6を解答とします。最後のxargs -I@は標準入力から受けた日付をdateの引数@ yesterdayの@の位置に放り込んでdateを実行するために用いています。

▼図4-3　関数nextwd

```
$ function nextwd(){ read d; date -d "$d $((7 - $(date -d $d +%w) + $(date -d $1 +%w))) day" +%F; }
```

▼図4-4　次週の火曜日が基準日から何日後かを求める

```
$ date -d 2017-09-18 +%w
1  ←月曜の曜日番号
$ date -d Tue +%w
2  ←火曜の曜日番号
$ echo $((7 - $(date -d 2017-09-18 +%w) ➘
+ $(date -d Tue +%w)))
8  ←1週間後から月曜の番号を引き、火曜の番号を足すと8日後に
```

▼図4-5　次週の火曜日が何日かを求める

```
↓2017年9月18日（月）から次週の火曜
$ date -d "2017-09-18 ➘
$((7 - $(date -d 2017-09-18 +%w) ➘
+ $(date -d Tue +%w))) day" +%F
2017-09-26
```

▼図4-6　解答

```
$ function nextwd(){ read d; date -d "$d $((7 - $(date -d $d +%w) + $(date -d $1 +%w))) day" +%F; }; ➘
echo 2017-09-18 | nextwd Tue | nextwd Fri | xargs -I@ date -d '@ yesterday' +%F
2017-10-05
```

問題5

うるう秒

上級 ★★★

出題、解説：上田　解答：eban

最近のうるう秒のあった日をなるべく多く列挙してください。

解答

図5-1が解答例です。まず、図5-2のワンライナーで「1月1日と7月1日の9時間後の1秒前」という意味の出力を作ります[注4]。9時間は世界標準時と日本の時間の時差です。

次に、タイムゾーンright/Japanを指定してdateに通すと、8時59分「60秒」というデータが混ざって出てくる（図5-3）ので、あとは解答例（図5-1）のようにgrepすればうるう秒が出てきます。タイムゾーンに対応するためのデータは「/usr/share/zoneinfo」にありますが、その下にうるう秒に対応した「./right」があり、この下のファイルを使うことになるそうです。

終わりに

今回は日付のデータ整形や計算を扱いました。dateutilsはほかにも便利な使い方がありそうなので、調べてみると良いかもしれません。次回はネットワークの問題（監視など）を扱います。SD

▼図5-1　解答

```
$ printf "%s 9 1sec ago\n" {1970..2017}-{01,07}-01 | ➘
TZ=right/Japan date -f- | grep :60
1972年  7月  1日 土曜日 08:59:60 JST
1973年  1月  1日 月曜日 08:59:60 JST
（..略..）
2017年  1月  1日 日曜日 08:59:60 JST
```

▼図5-2　「1月1日と7月1日の9時間後の1秒前」の出力を作る

```
$ printf "%s 9 1sec ago\n" {1970..2017}-{01,07}-01
（..略..）
2017-01-01 9 1sec ago
2017-07-01 9 1sec ago
```

▼図5-3　うるう秒対応のタイムゾーンを指定してdateに通す

```
$ （図5-2のコマンド） | TZ=right/Japan date -f-
（..略..）
2017年  1月  1日 日曜日 08:59:60 JST
2017年  7月  1日 土曜日 08:59:59 JST
```

注4）これまで、うるう秒の挿入は1月1日または7月1日に行われています。

RDBアンチパターン

PostgreSQLとMySQLの失敗と対策

Author 曽根 壮大（そね たけとも） ㈱はてな Twitter @soudai1025

第6回 ソートの依存

本連載では、開発の現場で発生しやすいリレーショナルデータベース(RDB)全般の問題をRDBアンチパターンとして紹介していきます。今回のアンチパターンの主人公は、発注履歴ページの表示遅延に悩む、発注管理の担当者。

ソートの依存

第6回目は「ソートの依存」という題名で、RDBの苦手なソートORDER BYについてのアンチパターンを紹介します。ソートはとても便利なRDBMSの機能ですが、パフォーマンス面で考えると、実はRDBが苦手とする分野です。今回はその理由と、それによって引き起こされる問題について説明します。

事の始まり

Aさん：このページ、表示が遅いなぁ……。

Aさんは会社の発注管理の担当者です。年度末に向け、今年の会計処理のために過去の発注履歴を確認しているところです。

Aさん：この発注履歴、最新のページはとくに問題ないけど、奥のページに行けば行くほど表示に時間がかかるな……。

Aさんは過去の発注履歴を見るためにページを遡(さかのぼ)っている途中ですが、ページが昔のものになればなるほど、表示に時間がかかるようです。

Aさん：これってなんとかならないのかなぁ。

悩んでるところに、情報システム部門のTさんが来ました。

Tさん：あー、これはソートの問題だね。
Aさん：ソート？
Tさん：ちょっと中身を見てみようか(**リスト1**)。やっぱり……。ページャの作りにも問題があって、そこも速度遅延の原因になってるね。順番に直していこうか。

今回のアンチパターン

今回のアンチパターンでは、RDBMSのソートの使い方に問題があります。この原因を知るためには、RDBMSのORDER BYやLIMIT、

▼リスト1　PostgreSQLで1年前の発注履歴を検索するクエリ

```
SELECT
  *
FROM
  "発注履歴"
WHERE
  "発注日時" > (now() + '-1 year');
ORDER BY
  "発注日時" DESC
LIMIT 100
OFFSET 5000;
```

OFFSETの挙動について知る必要があります。

リレーショナルモデルとソートのしくみ

ソートのしくみの前に、まず簡単にリレーショナルモデルについて紹介します。

リレーショナルモデル

RDBはRelational DataBase(リレーショナルデータベース)の頭文字を取っています。そしてその元となっている考え方がリレーショナルモデルです。リレーショナルモデルは集合を扱うデータモデルのことです。集合には次のような性質があります。

・重複がない
・実在する要素しかない(NULLがない)
・要素に順序がない

リレーショナルモデルのデータモデルは集合ですので、重複もNULLもソートもないのです。みなさん、アレ?と思うところがありますね。ソフトウェアとしてのRDBMS(Relational Database Management System:リレーショナルデータベース管理システム)には、重複もNULLもソートもあります。これはどういうことかと言うと、実際に必要なデータは多種多様で、リレーショナルモデルのみで表現することは難しいため、拡張されているのです。

RDBMSは、リレーショナルモデルよりも幅広くデータを扱えることで表現力が上がりましたが、やはりリレーショナルモデルがもとになっているため、リレーショナルモデルにない世界を扱うことは苦手です。つまり今回の例にあるようなソートは、リレーショナルモデルの外の世界の話ですので、パフォーマンスのボトルネックになりやすいのは必然なのです。

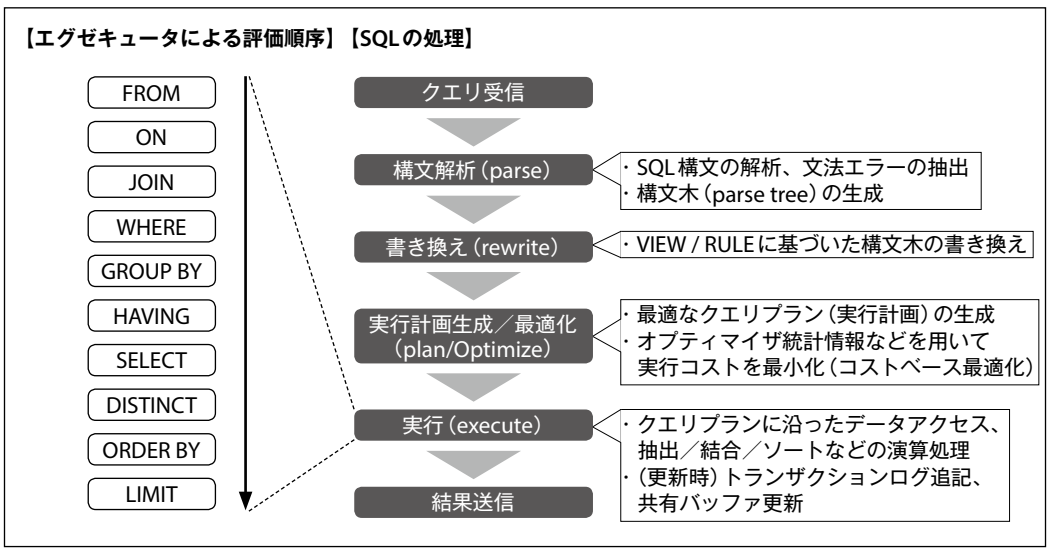
もしリレーショナルモデルについて興味が出た方は『理論から学ぶデータベース実践入門』[注1]が非常にお勧めですので、ぜひ読んでみてください。

ORDER BYのしくみ

リレーショナルモデルの世界にはソートがないことは、みなさんに伝わったと思います。ではRDBMSは、どのようにソートをORDER BYとして実装しているのでしょうか。RDBMSは**図1右**のように処理を進め、RDBMSのクエリ実行部分「エグゼキュータ」は**図1左**の順にSQL

注1) 奥野 幹也 著、2015年、技術評論社発行

▼図1 RDBMSとエグゼキュータ



▼図2　WHERE句狙いのINDEX

SQL：SELECT * FROM users WHERE 性別 = ‘男性’ ORDER BY 出身県id LIMIT 5

(1) 性別のINDEXを利用して検索

ユーザid	名前	性別	出身県id
1	テストA	男性	1
2	テストB	男性	1
3	テストC	男性	2
4	テストD	女性	3
5	テストE	男性	1
6	テストF	男性	3
7	テストG	男性	2
8	テストH	女性	1
9	テストI	女性	4
10	テストJ	男性	3
11	テストK	男性	4
12	テストL	男性	2
13	テストM	男性	2
14	テストN	女性	1

(2) 該当の結果(10件)を取り出し、それをソート

ユーザid	名前	性別	出身県id
1	テストA	男性	1
2	テストB	男性	1
3	テストC	男性	2
5	テストE	男性	1
6	テストF	男性	3
7	テストG	男性	2
10	テストJ	男性	3
11	テストK	男性	4
12	テストL	男性	2
13	テストM	男性	2

(3) ソート中に5件が確定したら結果を返す

ユーザid	名前	性別	出身県id
1	テストA	男性	1
2	テストB	男性	1
5	テストE	男性	1
3	テストC	男性	2
7	テストG	男性	2

を評価します[注2]。このように、すべてのデータを取り出してからORDER BYで並べ替え、最後にLIMITで必要なデータを切り分けます。

```
SELECT ← 3番目
  *
FROM ← 1番目
  users
WHERE ← 2番目
  id < 10000
ORDER BY name ← 4番目
LIMIT 100 ← 5番目
```

のように、データを取り出してからバラバラのデータを並び替えるため、ソートは高コストな処理ですし、実際のデータが大きくなればなるほど、さらに重い処理になります。

WHERE句狙いのINDEX

このようにORDER BYは、データが大きくなればなるほど重い処理になります。ですが事前にWHERE句を使えば、対象を絞り込むことができます。たとえば、1億件のレコードを並び替えるのと、それを1,000件に絞り込んでから並び替えるのとでは、同じORDER BY後に100件を取り出すとしても、処理時間が雲泥の差になることは歴然です。ですので、WHERE句を利用してデータを絞り込むことは重要ですし、さらにはWHERE句がINDEXを活用でき、データを十分に小さくできるのであれば、パフォーマンスは劇的に向上します(**図2**)。

ORDER BY句狙いのINDEX

WHERE句でデータを絞り込む優位性について説明しましたが、WHERE句が必ずしも最適解ではありません。前回(本誌2017年9月号)の「フラグの闇」でも紹介しましたが、WHERE句でINDEXを利用する列は、カーディナリティが少なく、データの値が偏っているとINDEXが有効に活用されません。

この場合に重要になるのが「ORDER BY句狙いのINDEX」と呼ばれる手法です。PostgreSQL、MySQLの標準的なINDEXの実装であるBTree INDEXは、データを「ソート済み」の状態で保存しています。勘の良い読者のみなさ

注2) 参考「PostgreSQL 9.6.3文書　SELECT」URL https://www.postgresql.jp/document/9.6/html/sql- select.html

▼図3　ORDER BY句狙いのINDEX

SQL：SELECT * FROM users WHERE 性別 = '男性' ORDER BY 出身県id LIMIT 5

(1) 出身県idのINDEXを利用して検索

ユーザid	名前	性別	出身県id
1	テストA	男性	1
2	テストB	男性	1
3	テストC	男性	2
4	テストD	女性	3
5	テストE	男性	1
6	テストF	男性	3
7	テストG	男性	2
8	テストH	女性	1
9	テストI	女性	4
10	テストJ	男性	3
11	テストK	男性	4
12	テストL	男性	2
13	テストM	男性	2
14	テストN	女性	1

(2) ソートされた結果(7件)を取り出しながら性別を評価

ユーザid	名前	性別	出身県id
1	テストA	男性	1
2	テストB	男性	1
3	テストC	男性	2
5	テストE	男性	1
7	テストG	男性	2
8	テストH	男性	1
14	テストN	男性	1

(3) 5件が確定したら結果を返す

ユーザid	名前	性別	出身県id
1	テストA	男性	1
2	テストB	男性	1
5	テストE	男性	1
3	テストC	男性	2
7	テストG	男性	2

ここでは7件を取り出して評価しているので図2の10件に比べて効率が良いうえ、ソートの処理が不要

んは気づきましたね？　そうです、対象のソート結果とINDEXが同じならば、INDEXから取り出せば良いのです(**図3**)。これをORDER BY句狙いのINDEXと言います。

図4は、件数10,317,168のテーブルで、ORDER BY句狙いのINDEXを使っていない場合と使った場合の実行計画です。rowsの項目を見るとわかるとおり、その差は一目瞭然です。SELECT * FROM users ORDER BY name LIMIT 10のクエリを実行すると、実行時間は11.51sec→0.01secと短縮できました。

ORDER BYでINDEXを利用すると、ソートの処理をすることなく、INDEXから該当のデータを順番に取り出すだけで済むので、高速に処理できます。ORDER BYにINDEXを利用した場合の強みは次の2つです。

・ソートの処理が不要になる
・評価数がLIMITの件数に達した時点で結果を返せる

ここまでPostgreSQLのORDER BY句狙いのINDEXについて説明しましたが、MySQLでも度々、INDEXを利用する際にORDER BYを"狙う"ことがあります。この話を@yoku0825さんのスライド資料『Where狙いのキー、order by狙いのキー』[注3]がよりわかりやすく説明しています。こちらはトランプを例に出したり、PerlでRDBのソートを表現したりした例があるのでぜひ見てみてください。またORDER BYやINDEXのしくみについては『SQLパフォーマンス詳解』[注4]がとても参考になります。



ページャの問題

実は今回のアンチパターンでは、ソート以外にももう1つ問題があります。それはページャの実装方法です。

例ではLIMITとOFFSETを使って表示の対象を選択しています。今回の事例では最新のページは高速に表示することから、発注日にはINDEXが使えていることが読み取れます。ではなぜページを遡っていくと重くなるのでしょ

注3）URL https://www.slideshare.net/yoku0825/whereorder-by
注4）URL http://use-the-index-luke.com/ja/sql/sorting-grouping

▼図4　ORDER BY句狙いのINDEX

```
そのまま実行
mysql> EXPLAIN SELECT * FROM users ORDER BY name LIMIT 10 ; ※実行結果を一部省略
+----+-------------+-------+------+------+---------+---------+----------+----------------+
| id | select_type | table | type | key  | key_len | rows    | filtered | Extra          |
+----+-------------+-------+------+------+---------+---------+----------+----------------+
|  1 | SIMPLE      | users | ALL  | NULL | NULL    | 9880259 |   100.00 | Using filesort |
+----+-------------+-------+------+------+---------+---------+----------+----------------+
1 row in set, 1 warning (0.00 sec)

ORDER BY句狙いのINDEX
mysql> EXPLAIN SELECT * FROM users ORDER BY name LIMIT 10 ; ※実行結果を一部省略
+----+-------------+-------+-------+------------+---------+------+----------+-------+
| id | select_type | table | type  | key        | key_len | rows | filtered | Extra |
+----+-------------+-------+-------+------------+---------+------+----------+-------+
|  1 | SIMPLE      | users | index | index_name | 182     |   10 |   100.00 | NULL  |
+----+-------------+-------+-------+------------+---------+------+----------+-------+
1 row in set, 1 warning (0.01 sec)
```

▼図5　アンチパターンのSQLの実行計画

```
mysql> EXPLAIN SELECT * FROM users ORDER BY name LIMIT 10 OFFSET 500; ※実行結果を一部省略
+----+-------------+-------+-------+------------+---------+------+----------+-------+
| id | select_type | table | type  | key        | key_len | rows | filtered | Extra |
+----+-------------+-------+-------+------------+---------+------+----------+-------+
|  1 | SIMPLE      | users | index | index_name | 182     |  510 |   100.00 | NULL  |
+----+-------------+-------+-------+------------+---------+------+----------+-------+
1 row in set, 1 warning (0.00 sec)
```

うか。それを紐解く鍵は**図5**の実行計画にあります。ここではkeyの項目のとおり、index_nameというINDEXを利用しています。さらにこのクエリはLIMIT 10のとおり、10件しか取り出しません。

ですがrowsの項目に注目してください。そうです！　510件も取り出しています。これはLIMIT 10 OFFSET 500のため、500行目まで順番に確認して、さらにそこから10件を取り出したため、全体としては500+10＝510件のデータを取り出しているのです。つまり、LIMIT 10 OFFSET 100000なら100,010件を取り出すことになります。これが今回のアンチパターンの大きな原因になっています。

✤ データは小さく、細かく取り出す

では、どのようにSQLを改善すれば良いでしょうか。ORDER BYを速くするには、

・データを小さくする

・INDEXを活用する

の2点が重要です。ですがINDEXを活用できたとしても、全体の件数が多くなるとうまくいきません。INDEXは第4回（本誌2017年8月号）の「効かないINDEX」でもお話ししたとおり、全体のデータの1割から、多くても3割程度の結果の場合でなければ活用されません。つまり、OFFSETで指定したデータ範囲がそれを超えると有効活用できないということになります。

ではどうすれば良いでしょうか？　この場合、WHERE句で絞る値を追加することで解決するテクニックがあります。

```
SELECT
  *
FROM
  "発注履歴"
WHERE
  "発注日時" > (now() + '-1 year');
  AND id > :前の表示ページの最後の行のid
ORDER BY
  "発注日時" DESC;
LIMIT 100
```

このように、OFFSETではなく表示したい発注日時の最後の行を指定してさらに絞り込んでいます。アプリケーション側で「次のページ」として渡すのはOFFSETの値ではなく、最後に表示された行のidにすることですばやく絞り込めます。

これは、「発注日時が順にINSERTされるのであればidも発注日時と同様に並んでいる」ということを利用しています。また、発注日時ではなくidで絞り込みしているのは、発注日時は必ずしもUNIQUEではないため、場合によっては発注日時がページとページの間でかぶり、条件によって表示されなくなることがあるためです。この手法には次のメリットがあります。

- データ量が増えてもINDEXを活用できるため高速
- ページ数が深くなってもOFFSETを利用しないため取得行が肥大化しない

また、SELECTの指定項目がidと発注日時のみの場合は、MySQLの場合は「カバリングインデックス」、PostgreSQLの場合は「インデックスオンリースキャン」が期待できます。INDEXに含まれているデータのみが必要な場合は実際のテーブルデータにアクセスせずに結果を返せるため、より高速です。

このようにORDER BYやINDEXのしくみを知り、SQLを工夫することでパフォーマンスを何十倍にもすることができます。ただしこの手法は、次のような場合には使えません。

(1)ORDER BYの結果がidの順番と関係なく、またUNIQUEな値ではない

たとえばnameなど、UNIQUEな値でなく、idが順不同の場合です。ただし、`ORDER BY name, id DESC`のように、idでもソートすることで改善する場合もあります。

(2)UNION/GROUP BY/HAVINGが使いたい

UNIONやGROUP BYやHAVINGを利用した場合、結果にidが使えないケースがあります。この場合はPostgreSQLの場合はGENERATE_SERIES()などで無理矢理に連番を作ることができますが、GROUP BYの結果などはINDEXが当然ないため、ORDER BY句狙いのINDEXなどは使えません。UNIONなどが高速に動作し、データ量が少ない場合は大きな問題にはなりませんが、OFFSETの例と同様に、データ量が大きくなるとORDER BY狙いのINDEXが使えない問題が顕在化します。また大前提として、GROUP BYなどとORDER BYは相性が良くありません。そのため、GROUP BYとORDER BYが組み合わさったケースでは、集計結果のSummaryテーブル(PostgreSQLの場合はマテリアライズド・ビューでも良い)を作り、そこにINDEXを貼ることを検討しましょう。

ページャの難しさ

idを指定することでページャを高速化することを紹介しました。JOINや複雑なWHERE句を必要とする場合はテーブルスキャンが発生しやすく、それに対してORDER BYを利用するため、パフォーマンスのボトルネックになりがちです。また、データは少しずつ増えていくので、最初は表示に問題がなくても時間の経過とともに問題が顕在化してくるのもページャの問題点の1つです。

ですが、ページャはRDBのパフォーマンス以外にも多くの問題を抱えています。たとえば、3ページから4ページ目にアクセスする途中に1ページ目のデータが削除されると、表示される行が変わるため、表示されずに飛ばされる行が発生します。このように、ページャには技術的にいろいろな課題がありますので、実装が悩ましい機能の1つです。

大きなデータをソートしたいときは？

結論を言えば、RDBMSの機能を使わずに実装することになります。一般的な方法は次の3つのいずれかでしょう。

(1)アプリ側でソート

RDBMSにソートをさせないため、RDBの負荷を下げることができるメリットがあります。たとえば、フロントエンド側からデータの一覧をREST APIで取得し、それをJavaScript側でソートさせて表示させる方法です。表示に合わせて複雑なソートがしたい場合や、同じデータに対してリアルタイムかつ頻繁にソート条件を切り替えたい場合などで有効なケースです。最近はネイティブアプリなど、クライアント側の環境がリッチになってきたため有効な方法の1つですし、ソートの処理を分散化できるメリットもあります。しかし、データ一覧をアプリにすべて渡す必要がありますし、データサイズが大き過ぎると通信がボトルネックになることもあるので注意しましょう。

(2)ソート済みの結果をキャッシュして利用

ソートの処理が決まっており、結果が変更されにくいデータなどはキャッシュが有効です。たとえば、郵便番号や市町村の住所などが該当します。ただし、キャッシュは更新頻度が高いデータでは使い物になりませんし、アーキテクチャの複雑度が上がりますので、使わずに済むのであれば使わず対応するのが良いでしょう。またデータをキャッシュするのではなく、表示結果が同じ場合はHTML自体をVARNISH[注5]などのHTTPアクセラレータでキャッシュしたり、CDNを使ってJSONをキャッシュしたりする手法もあります。

(3)NoSQLなどを利用してソート

近年ではこの方法が一般的です。RDBMSはとても便利なソフトウェアであることは間違いありません。とくにデータが少なく、システムとしてシンプルな場合は、RDBMSの機能だけでシステムを実現することで運用もシンプルになることが多々あります。しかし、RDBの苦手な処理を行うと、パフォーマンスに大きな課題が生まれます。このような場合は、NoSQLなどの特定用途に特化した製品を利用して実装することも検討しましょう。適材適所に選択肢を選ぶこともエンジニアの腕の見せ所です。

今回のアンチパターンのポイント

ORDER BYでINDEXを使うことを始め、RDBMSのソートについてお話しました。では実際に業務でうまく活用するためにどうするか？と言われた場合の答えは「しっかりと実行計画を見る」ということです。

注5）URL https://varnish-cache.org

RDB TIPS 実行されないとわからないORDER BYの問題

ORDER BYを利用したクエリが急激に遅くなる瞬間があります。それは、メモリ内ですべてを処理できなかったときです。PostgreSQLもMySQLも、ソートの処理はメモリ上で実行できる場合にはクイックソートを利用しますが、それ以上のサイズになると、ソート結果をファイルに書き出す外部ソートになります。ORDER BYの外部ソートは処理が遅くなる典型的なパターンの1つで、次の2つの問題があります。

- データ量が成長することで当初は問題がなかったクエリがある日突然外部ソートになる
- クイックソートか外部ソートかは実行されるまでわからない

とくに「実行されるまでわからない」は、RDBMS側のモニタリングで予測することが難しいパターンの1つです。そのため、スロークエリログをモニタリングしてチェックしたり、外部ソート時に生成されるファイルを監視したりして、いち早く問題に気づけるようにすることが大切です。

たとえば、ORDER BY句狙いが良いか、それともWHERE句狙いが良いかは、中に入っているデータによって変わります。オプティマイザは今の情報しか知らないため、未来にどんなデータが入るかわかりません。そのため、場合よっては不適切な実行計画を選ぶことがありますし、実行計画がある日変わることもあります。だからこそ、実行計画をしっかりと見ることが大切ですし、実行計画を見るとこれまでに学んだRDBMSの挙動が見えてくるはずです。

最近はMySQL Workbench[注6]やpgAdmin 4[注7]など、実行計画をグラフィカルに表示してくれるツールも出ています。これらを活用してぜひ実行計画を見てみてください。既存のシステムでも、予想外の発見が見つかるかもしれません。

注6）URL https://www.mysql.com/jp/products/workbench
注7）URL https://www.pgadmin.org

次回のRDBアンチパターン

今回のRDBアンチパターンはいかがでしたでしょうか？　ソートの依存は現場でもよく見られるアンチパターンです。また、改善するためにはアプリケーションの影響が多く、あとから修正するのがたいへんなのも特徴のアンチパターンです。みなさんも初期設計の段階で、ソートについてはしっかりと検討・実装してください。

それでは次回は久々にデータに関する恐ろしいアンチパターンです。次回の「隠された状態」もお楽しみに！SD

RDB TIPS　RDBを補う存在、Redis

RDBの苦手な処理はNoSQLに任せるのも選択肢の1つと本編で紹介しました。その中でもRedisは、RDBMSと苦手なところをカバーし合うソフトウェアです。Redisは非常に高速に動作するインメモリ型キー値データ構造ストアです。

- **非常に高速に処理を行うことができる**
  すべてのデータをメモリ上で処理するため非常に高速（永続化することも可能）
  豊富なデータ型とそれに対する演算方法が多彩
- **簡単にレプリケーションを行うことができる**
  非同期レプリケーションによって可用性の向上と参照のパフォーマンス向上を行える

以上のような特徴からRedisは、キャッシュ、セッション管理、キュー、ソートといった場面でよく利用されます。

このようにRDBの苦手な部分を高速に処理できるため、RedisはRDBMSと非常に相性が良いです。とくにSorted Set（ソート済みセット型）を使うと、高速にソート済みデータを検索したり、取得したりすることができます。これは、ORDER BYがINDEXから高速にデータを取得できることに似ており、それ専用の型を持っているRedisの強みです。

しかし、Redis自体は永続的にデータを扱うことは苦手ですし、保存できる容量が制限されており、クラッシュセーフではありません。その点もRDBMSとの住み分けになっており、トランザクションが必要な課金系のデータはRDBMSに保存し、その結果はRedisに保存して参照はRedisから行うといった構成はよく行われます。

Hadoopのような大量データの分散処理とはまた違ったアプローチによる、RDBMSとの住み分けができるRedisは、みなさんの設計にとても役立つ選択肢の1つです。今まで使ったことがない方は、ぜひこの機会に調べて覚えてみてください。

presented by
Japan Android Group
http://www.
android-group.jp/

コミュニティメンバーが伝える Androidで広がるエンジニアの愉しみ

第19回 セーラー服を彩るフルカラーLEDを制御する

Androidは世界で出荷される約9割のスマートフォンに搭載される標準OSです※。そのため、多くのAndroidアプリが開発され続けており、そして多くのエンジニアが活躍しています。Androidで広がる新しい技術に魅了されたエンジニアが集うコミュニティもあり、そこでは自分が愉しむための技術を見つけては発信しています。その技術の一幕をここで紹介します。

※Gartner Worldwide Smartphone Sales to End Users by Operating System in 3Q16

日本Androidの会秋葉原支部
支部長 鈴峰 きり(すずみね きり)
小暮 敦彦(こぐれ あつひこ)
林 博(はやし ひろし)

今回はAndroidアプリからBluetooth通信経由で、LEDの色と輝度を自由に制御する方法について紹介します。

紹介するハードウェアとソフトウェアは、テクノロジをファッションに応用する取り組みの中で生まれた「電飾セーラー服」のために開発しました。実のところ、秋葉原のメイドさんに着てほしい一心で、おじさんたちが仕事以上に頑張りました。

この作品は実際に、ウェアラブルデバイスファッションショー(2017年6月8日、幕張メッセ「LEJ Night」[注1])において秋葉原のメイドさんが着用しました(**写真1、2**)。

誌面の都合もあり、この先の解説は次の補足資料を手元にダウンロードしてお読みください。

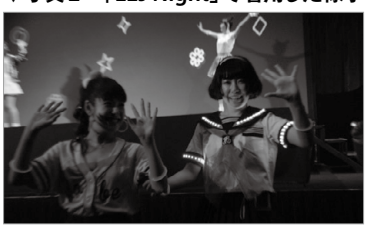
①マイコンサンプルプログラム
https://github. com/hhiro-GSX1100/LedTape Controllerの「LedTape.cpp」
②マイコン解説
http://techmode.co.jp/sd_jag.pdf

注1) http://www.f2ff.jp/dsj/2017/lej/#lej_night

▼写真1 電飾セーラー服



▼写真2 「LEJ Night」で着用した様子

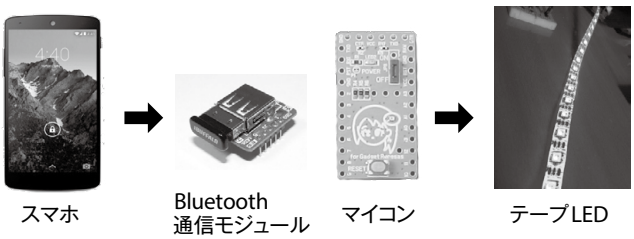


システム構成

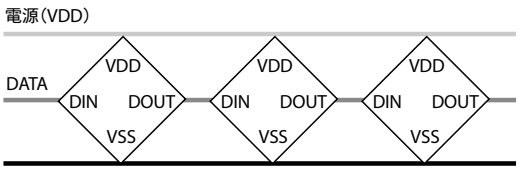
制作するシステムの構成は、LEDと、これをシリアル通信で制御するコントローラ(マイコン)、およびマイコンにBluetooth経由で指令を出すAndroidアプリから成り立っています(**図1**)。

AndroidアプリはBluetooth SPP(Serial Port Profile)によりBluetooth通信モジュール

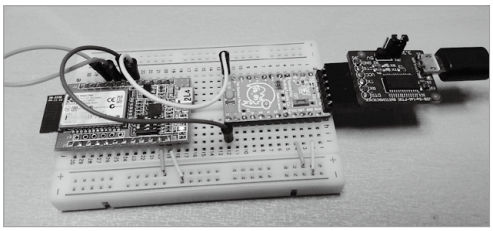
▼図1 電飾セーラー服のシステム構成



▼図2　WS2812BのLED制御線概略図



▼写真3　コントローラ部



【必要品】ブレッドボード、GR-KURUMI、RN-42使用Bluetooth無線モジュール評価キット(秋月電子)、FTDI USBシリアル変換アダプタ(5V/3.3V切り替え機能付き)(スイッチサイエンス)
【注意点】電源はFTDI USBシリアル変換アダプタの3.3Vを使用します。5V/3.3Vの切り替えが可能でも、ロジックだけのものがあるので注意してください(スイッチサイエンスのアダプタは、VCCが3.3Vになります)

越しにコントローラとの間で双方向通信を行い、その指示データに従いマイコンがシリアル通信でテープLEDを制御します。

コントローラにはマイコンプログラミングが必要となります。マイコン側のプログラミングと配線の基礎については前記②資料「マイコン解説」を参照してください。

LED

LEDにはWS2812Bという制御回路内蔵のものを用いています。このLEDは、電源(VDD)、Ground(VSS)および信号線(DIN)の3線(マイコンとは実質DINの1線)の接続で、複数のLEDをコントロールできるものとなっています(**図2**)。複数接続する場合は、信号線のDINとDOUTをdaisy-chainにより結線することで実現できます。

開発元のWorldsemi Technologyの資料によれば、1LEDあたり24bit(8bit×3色)の信号を与え、RES(Reset)信号を送ることにより各色の輝度を8bit階調で制御できます。

▼リスト1　GR-KURUMIの電気信号を生成するコード

```
// 1値(bit)を出力するコード
inline void LedTape::code_1(){
    *m_Px |= m_On_u8Bit;  // 出力をHIGHに設定
    wait_T1H();
    *m_Px &= m_Off_u8Bit; // 出力をLOWに設定
    wait_T1L();
}
// 0値(bit)を出力するコード
inline void LedTape::code_0(){
    *m_Px |= m_On_u8Bit;  // 出力をHIGHに設定
    wait_T0H();
    *m_Px &= m_Off_u8Bit; // 出力をLOWに設定
    wait_T0L();
}
```

コントローラ

コントローラは、おもにマイコンとBluetoothモジュールで構成しており、マイコンは、Arduino Pro Miniピン互換のGR-KURUMI[注2]を用いています(**写真3**)。GR-KURUMIを用いた理由は、ピンクで可愛い秋葉原サブカルのイメージであることと、何より高性能であることが主たる要因です。もちろん、Arduino Pro Miniでも再現できます。Arduinoの場合はAdafruit社のNeoPixel Library[注3]を使用できますので、前記LEDの項で説明した電気信号の生成を考える必要がなくなり、ソフトウェアの実装が容易になります。

リスト1にGR-KURUMIの電気信号生成部分のプログラムを示します。ここからはマイコンのサンプルプログラム「LedTape.cpp」をあわせてご覧ください。

m_Pxは出力ポートレジスタです。m_On_u8Bitは、信号を出力するマイコンのピンの出力をHIGHにするためのマスクビットであり、OR演算することにより指定のピンがHIGHになります。また、m_Off_u8Bitは、信号を出力するマイコンのピンの出力をLOWにするため

注2) http://gadget.renesas.com/ja/product/kemuri.html
注3) https://github.com/adafruit/Adafruit_NeoPixel

のマスクビットであり、AND演算することにより指定のピンがLOWになります。

続いて**リスト2**にLEDの色の信号を生成するコードを示します。

ptrはLED設定値(バイトの配列)のポインタです。このポインタから1バイトの値を取り出し、変数valに格納。これとビットマスクとのAND演算により、当該ビットが1か0かを判断し、**リスト1**に示した「1」「0」の信号を出力する関数を実行しているわけです。

また、noInterrupts()は割り込みを禁止するための関数であり、これにより、実行途中で割り込みによる遅延を回避しています。マスクビットをシフト演算(>>)によりループしておらず、冗長な書き方となっていますが、これは演算にかかる遅延時間について、LEDのData transfer timeに与える影響が無視できないからです。最終的にこの信号生成部分のコードは、オシロスコープにより時間計測して決定しました。

さて、AndroidとのBluetooth接続は、Bluetoothシリアルモジュールをマイコンと接続することにより、Androidとマイコン間でシリアル通信することに帰結します。したがって、Android側から送られたコマンド(文字)により動作をコントロールするように、マイコンのプログラムを書けば良いのです。

マイコンのシリアル通信のプログラムもライブラリを用いて比較的容易に開発ができます。ただし、Bluetoothシリアルモジュールとマイコンのロジック電圧は合わせるようにしてください(マイコンが3.3Vなら、Bluetoothシリアルモジュールも3.3Vのものを使用するか、レベル変換モジュールにより変換します)。

▼リスト2　LEDの色の信号を生成するコード

```
    noInterrupts();
 ...略...
    for(int i = 0; i < numBytes ; ++i){
        val = *ptr;
        if(ptr != ptr_end) ptr++;

        if(val & 0x80){
          code_1();
        }else{
          code_0();
        }

        if(val & 0x40){
          code_1();
        }else{
          code_0();
        }
 ...略...
    }
    interrupts();
```

Androidアプリ

Bluetoothで通信するプログラムとしては、Android Studio(AS)のサンプルにBluetooth Chatというプログラムがあり、これを用いれば容易です。ASで、[File]－[New]－[Import Sample]を選択。「Bluetooth」で検索すれば出てきます。

注意点として、そのままでは接続できず、UUIDをSPP(Serial Port Profile)用の文字列(00001101-0000-1000-8000-00805F9B34FB)に変更する必要があります(**図3**)。「Bluetooth Chat UUID」でWeb検索すれば、参考となる記事がいくつも見つかると思います。

コンパイルしてAndroid端末にインストールすれば、すぐに使用できます。なお、我々が作成したアプリ「FullColorLedTapeController」をGoogle Playで公開しています[注4]。

サンプルの動作

Bluetooth Chatをそのまま用いて、GR-KURUMIと接続テストするプログラムを**リスト3**に示します。

GR-KURUMIにWebコンパイラで**リスト3**をコンパイルしたプログラムを書き込み、TeraTermなどでFTDI USBシリアル変換アダプタのCOMポートに接続します(ボーレートは9600)。Bluetooth Chatを起動し、Blue

注4) https://play.google.com/store/apps/details?id=com.application.hiro.ledtapehairband&hl=ja

tooth接続すれば双方向で文字のやりとりが可能です。

また、Bluetooth Chatから文字'a'を送信すると、GR-KURUMIのLEDが赤に点灯し、ほかの文字を送信すると消灯します。

以上のとおり、比較的簡単にAndroidからBluetoothを用いてコントロールできることがおわかりいただけると思います。

日本Androidの会秋葉原支部では、マイコンとWebの橋渡しをするAndroidアプリやWebアプリケーションを作成し、インタラクティブ要素を持つアイテムを開発しています。たとえば、

「メイドさんが着たセーラー服の色をみんなのスマホでコントロール」

「メイドさんの吐息を地球の裏側に伝送する装置」

「メイドさんの鼓動を体感するシステム」

など、楽しい取り組みを実践しています。SD

▼図3　BluetoothChatService.javaのUUIDをSPPに変更（この例では「MY_UUID_SECURE」および「MY_UUID_INSECURE」の両方を変更している）

BluetoothChatService.java ×

BluetoothChatService | MY_UUID_SECURE

```
46      private static final String NAME_SECURE = "BluetoothChatSecure";
47      private static final String NAME_INSECURE = "BluetoothChatInsecure";
48
49      // Unique UUID for this application
50      //private static final UUID MY_UUID_SECURE =
51      //        UUID.fromString("fa87c0d0-afac-11de-8a39-0800200c9a66");
52      private static final UUID MY_UUID_SECURE =
53              UUID.fromString("00001101-0000-1000-8000-00805F9B34FB");
54      //private static final UUID MY_UUID_INSECURE =
55      //        UUID.fromString("8ce255c0-200a-11e0-ac64-0800200c9a66");
56      private static final UUID MY_UUID_INSECURE =
57              UUID.fromString("00001101-0000-1000-8000-00805F9B34FB");
58
59      // Member fields
```

▼リスト3　GR-KURUMIのテストプログラム

```
#include <Arduino.h>

#define LED_RED    22 // LOW active

int val = 0;
char ch;

void setup() {
    Serial.begin(9600);
    Serial2.begin(9600);
    pinMode(LED_RED,OUTPUT);
}

void loop() {
    while(Serial2.available()){
        ch = Serial2.read();
        Serial.write(ch);
        if(ch=='a'){
            digitalWrite(LED_RED,LOW);
        }else{
            digitalWrite(LED_RED,HIGH);
        }
    }
    while(Serial.available()){
        Serial2.write(Serial.read());
    }
}
```

COLUMN

支部長　鈴峰きりのコメント



日本Androidの会秋葉原支部は、異素材を組み合わせ新しい価値を創造することをテーマに約5年間ファッション×テクノロジの活動をしてきました。

ファッション教育最大手である文化服装学院とインターンシップ制度を組み、光るセーラー服や光ファイバー布を使った着物、電脳アクセサリーの製作など、メンバーが力を合わせて手探りでものづくりをしています。

ファッション系の学生とIT系技術者という普段接することのない異分野の人同士が関わることで、新しい発見や貴重な学びが生まれます。

私自身、秋葉原支部の活動に参加する前までは、ファッションとテクノロジは対極にあると思っていました。しかし、さまざまな活動を通し、ものを作る中で両者が融合することに大きな可能性を感じています。

近年では「ファッションテック」という造語ができるくらい、ファッション×テクノロジのジャンルはアパレル業界からの注目度も高く、世界中でファッション性の高いIoT／ウェアラブルアイテムが開発されています。

この活動の延長で、今回「ウェアラブルデバイスファッションショー」を実現しましたが、いつかはもっと大きな舞台でファッションとテクノロジを融合したショーやイベントをやりたいと思っています。

秋葉原支部の活動に少しでも興味を感じた方がいましたら、参加や見学をお待ちしています！　一緒に手を動かしてファッションテックの未来にイノベーションを起こしましょう！

（協力：メイド喫茶 橙幻郷　有村詩音、鈴峰きり）

一歩進んだ使い方のためのイロハ

Vimの細道

第22回

foldでテキストを折り畳む（前編）

mattn
twitter:@mattn_jp

ソースコードの折り畳みは、IDEでは一般的な機能ですが、Vimでも標準で使うことができます。今回と次回とで、そのVimの折り畳み機能「fold」を解説します。前編では、マーカーやインデントを使った基本的な折り畳みを扱います。

折り畳みのメリット

テキストエディタはソースコードや文章を書くものです。ソースコードであれば関数や処理を、文章であれば構造化された章や節を書きます。

構造化されたテキストはおおよそブロックごとに記述されており、ブロックの先頭に章題を記述するかと思います。これら章題は見出しとして一覧することができ、ユーザはそれを目次として扱えるようになります。一般的にそういった見出しはアウトラインという機能により実現されますが、その実現方法の1つに折り畳みという機能も存在します。Visual StudioなどのIDEでも、メソッドの中身を折り畳むことができます。

Vimにも折り畳み機能が存在します。それはfoldと呼ばれ、あらかじめ用意された識別子を使って本文を折り畳んだり、折り畳みのルールを作成して独自の折り畳みを実現したりすることもできます。テキストだけで実現できる折り畳みとしては、Vimは強力過ぎるほどの機能を持っています。今月号と来月号に分けて、この強力なfoldの機能について説明したいと思います。

基本的な折り畳みをマスター

Vimの折り畳み方法は何種類か用意されており、foldmethodというオプションにて変更できます（**表1**）。

手動による折り畳み

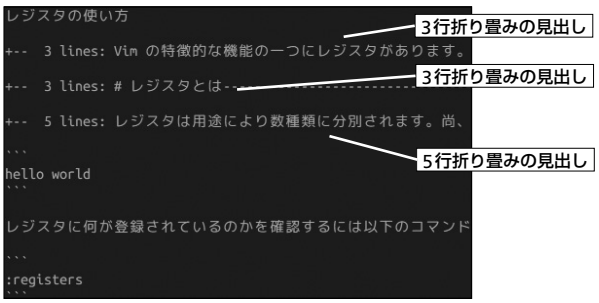
手動による折り畳みは、折り畳みたい開始行から終了行までを選択し、:'<,'>foldを実行します。ビジュアル選択した状態で:をタイプすると:'<,'>がはじめから入力されているので、続けてfoldを入力してください。何回か繰り返し実行すると、**図1**のような表示になります。

選択した状態からzfをタイプしても同様に折り畳むことができます。またノーマルモードから折り畳む場合は、zfのあとにモーションを指

▼表1　foldmethodオプションの値とその説明

foldmethodの値	説明
manual	手動による折り畳み
marker	マーカーによる折り畳み
indent	インデントレベルによる折り畳み
syntax	シンタックスハイライトによる折り畳み
diff	変更のない個所を折り畳み（diffモード専用）
expr	foldexprオプションによるプログラマブルな折り畳み

▼図1　手動による折り畳み



▼図2　マーカーによる折り畳み



定することもできるので、たとえばzf3jを実行すると、現在行から3行下の行までが折り畳まれます。また、たとえばXMLファイルで次を実行すると、タグ<foo>から</foo>までの行を折り畳むことができます。

```
:g/<foo>/;/<¥/foo>/fold
```

※同一行にある場合は折り畳まれません。

折り畳んだ行を含んでさらに折り畳むことで、ネストした折り畳みを作成することもできます。

折り畳みを開くには、その折り畳まれた行でzdをタイプします。ネストされた折り畳みを再帰的に開くにはzDをタイプします。

折り畳まれた行には、複数行あるテキストの先頭行がプレビューで表示されます。折り畳まれた行は1行のように扱うことができるため、たとえばその行をddで削除し、別の個所でペースト(p)することで、複数行のテキストの移動ができます。

foldを使っていると、1つ気付くことがあるかと思います。折り畳まれたテキストを含むファイルを保存して再度開いたとしても、折り畳まれた状態が保持されないのです。もちろんテキストですから、何かしらのマーカーでもない限りはそこに折り畳みがあったかどうかが判別付きませんので、当然ですね。

▼リスト1　C言語のソースコードを折り畳み

```
/* インクルード {{{ */
#include <stdio.h>
#include <stdlib.h>
/* }}} */

/* データベース関連 {{{ */
... データベース関連関数
/* }}} */

/* ファイル入出力 {{{ */
... ファイル入出力関数
/* }}} */

/* vim:set foldmethod=marker: */
```

マーカーによる折り畳み

Vimには、テキストの中の特定の識別子を見つけて、自動で折り畳むしくみが備わっています。デフォルトでは{{{と}}}がマーカーと呼ばれる識別子として扱われ、オプションfoldmethodをmarkerに設定することで折り畳まれます。

このfoldmethodはモードライン[注1]で指定することができ、ファイルの先頭や末尾にコメント行として書いておくことで、ファイルを開くと同時に折り畳んだ状態で表示することができます。たとえば次のテキストをファイルに保存して開くと、折り畳まれた状態で表示されます（**図2**）。

```
{{{
 このテキストは折り畳まれます。
}}}

// vim:foldmethod=marker
```

ソースコードであれば、カテゴリごとに関数を折り畳んでおくと構造がわかりやすくなります。たとえば、C言語のstatic関数と外部から参照される関数を**リスト1**のように分けておくと便利かもしれません。

注1） setを用いない省略記法などモードラインの書式にはいくつか種類があります。またモードラインを有効にするには、設定ファイルに:set modelineが必要です。

◆マーカーをネストする

マーカーはネストすることができます。まずリスト2のソースファイルを開くと、次のように折り畳まれています。

```
+--  3 lines: global variables ---
+--  8 lines: functions ----------
```

「functions」の折り畳みを開くと、次のように関数が折り畳まれているのがわかるかと思います。

```
+--  3 lines: global variables ---
/* functions {{{1 */
+---  3 lines: funcA() -----------
+---  4 lines: funcB() -----------
```

}}}は次に現れるマーカーまでを折り畳み範囲とする場合には、明示的に}}}を記述する必要はありません。

◆モードラインが使えない場合

モードラインが使えない状況、たとえば業務のソースコードにVimのモードラインのようなものを含められないといった場合には、ファイルタイプを使ってこのオプションを設定します。リスト3をvimrcに追加しておくと、以降はファイルタイプcppのファイル(C++)を開いたときに{{{から}}}までが折り畳みの対象となります。

事情によりマーカーすら記入することができない場合は、後述するsyntaxによる折り畳みを使うこともできます。

インデントによる折り畳み

インデントのレベルによる折り畳みです。同一インデントレベルを含む並んだ行が折り畳まれます。ここで注意しておいてほしいのは、これはツリー状に折り畳むものではないということです。

リスト4の❷の位置でzcをタイプすると、2つ並んだ「レベル3」が折り畳まれますが、❹でzcをタイプしても折り畳まれません。あくまで、「同一インデントレベルを含む並んだ行」が折り畳まれるので、同じインデントレベル、もしくはそれ以上のインデントレベルの行が並んでいなければ折り畳まれません。同様に、「レベル1」も同一のインデントレベルが存在しませんので、折り畳まれません。❶や❸や❺でzcをタイプすると「レベル2」が1行に折り畳まれます。

ただ、ツリー状に折り畳む方法がないわけではありません。そちらは次号で紹介したいと思います。

シンタックスハイライトによる折り畳み

各ファイルタイプのシンタックス定義に基づいて折り畳みが行われます。各ファイルタイプごとに折り畳まれ方が異なります。たとえばRubyであれば、リスト5の①②③の3つが折り畳まれます。このほか、複数行の正規表現や複

▼リスト2　ネストされたマーカー

```
/* global variables {{{1 */
int varA, varB;

/* functions {{{1 */
/* funcA() {{{2 */
void funcA() {}

/* funcB() {{{2 */
void funcB() {}

/* vim:set foldmethod=marker: */
```

▼リスト3　ファイルタイプcppのファイルを折り畳む設定

```
augroup MyCppFoldmethod
  au!
  autocmd FileType cpp setlocal foldmethod=marker
augroup END
```

▼リスト4　インデントレベルによる折り畳み

```
レベル1
  レベル2 ❶
    レベル3 ❷
    レベル3
  レベル2 ❸
    レベル3 ❹
  レベル2 ❺

# vim:set foldmethod=indent:
```

▼リスト5　シンタックス定義による折り畳み（Rubyの場合）

```
def foo(x)
  x + 1
end

[1,2,3].map do |x|
  x + 1
end

x = [1,2,3].map {|x|
  x + 1
}

# vim:foldmethod=syntax
```

①
②
③

数行のバッククオート、複数行のパーセント記法（%qや%w）などが折り畳まれます。

C#であれば、#regionから#endregionの折り畳みもサポートしています。**リスト6**ではメンバ変数とメンバ関数で折り畳まれ、さらにメンバ関数の折り畳みの中にプライベート関数とパブリック関数が折り畳まれます。

◆　◆　◆

今回はVimの強力な折り畳み機能の使い方を紹介しました。使い方しだいではさまざまな場面で活用することができ、文書やソースコードの構造を見やすくすることができます。ぜひ一度使ってみてください。 次号ではfoldexprを使った独自の折り畳みを作る方法を紹介したいと思います。SD

▼リスト6　シンタックス定義による折り畳み（C#の場合）

```
using System;

class Program
{
  #region メンバ変数
  int v = 0;
  #endregion

  #region メンバ関数
  #region プライベート関数
  private void doPrivateThing() {
    Console.WriteLine("value is {0}", v);
  }
  #endregion

  #region パブリック関数
  private void doPublicThing() {
    doPrivateThing();
  }
  #endregion
  #endregion

  #region メイン関数
  static void Main(string[] args) {
    new Program().doPublicThing();
  }
  #endregion
}

// vim:foldmethod=syntax
```

Vim日報

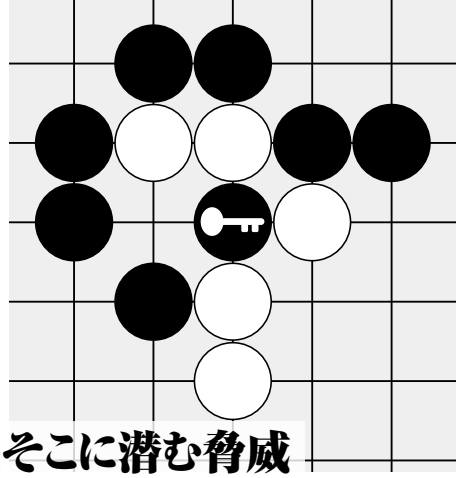
なかなか直らないバグ

Vimの開発者コミュニティであるvim-jpでは、さまざまな問題を解決し、Vimの開発グループであるvim-devへ橋渡しを行います。その中には、日本人のようにマルチバイト文字を使うユーザでないと発生しないバグも多くあります。そういったバグの中には、Vimの作者であるBram Moolenaar氏を説得するのに時間がかかるものもあります。それを重大なバグとして認めてくれないのです。マルチバイト圏の人間にとっての重大なバグ修正パッチも、時には流れて消えてしまうことがあります。

Windowsのvim.exeは終了後に起動前の画面を復帰しているのですが、その際に画面にユニコード文字が表示されていた場合に、それが正しく復帰できていませんでした。このバグはずいぶん昔からありました。原因はWin32ワイド文字APIを使っていなかったことにあるのですが、英語圏のBram氏には、これが重大な問題だとは思えなかったようです。vim-jpのk-takata氏と筆者で「なぜマージしないの？」という返信を何度も行い、つい先日、ようやくマージされました。小さなバグ修正だったかもしれませんが、大きな偉業を達成したと思っています。これは正直言って粘り勝ちでした。

セキュリティ実践の基本定石

すずきひろのぶ
suzuki.hironobu@gmail.com



みんなでもう一度見つめなおそう

【第四八回】ソフトウェアのサプライチェーンとそこに潜む脅威

「危険なサイトには近づかない」とか「怪しいソフトウェアはダウンロードしない」などの呼びかけをよく見かけます。最初から怪しいソフトウェアをダウンロードして使うような人は、本誌読者にはいないと思いますが、それまで信頼して使っていたソフトウェアが突然マルウェアに化けることが万が一あったならば……それは悪夢でしょう。

ソフトウェア管理とサプライチェーン

ここで言うソフトウェアのサプライチェーンとは、利用するソフトウェアの提供とそのメンテナンスも含めたアップデート・サポートの環境のことです。現在ではソフトウェアをインストールしたあと、そのソフトウェア本体やシステムのソフトウェア管理ツールがアップデートを管理するケースが多くなっています。もしこのような機能が提供されていなければ、利用しているソフトウェアの脆弱性管理のために、私たちは日課のようにソフトウェアの提供元の情報を見まわらなくてはなりません。

今はそのようなことをしなくても自動化されているのでたいへん便利です。たとえばスマートフォンなどは明確で、iPhoneであればApp Store、AndroidであればGoogle Playストアからアプリをダウンロードし、その後のアプリのアップデートはデフォルトでは自動化されています。そのため、ユーザがいちいちメンテナンスに気を配らなくても適切にアップデートされています。このようなしくみがあるので、多様なアプリが使えるスマートフォンでもユーザの裾野（すその）を広げることができました。もしこれがソフトウェアのメンテナンス知識が必須であるような状況であれば、今日のスマートフォンの普及状況とは違うものになっていたでしょう。

筆者はGNU/Linuxを常用しています。ノートPCとデスクトップPCで利用しているディストリビューションはUbuntuで、サーバにはCentOSを使っています。これらの上で使っているソフトウェアはすべてオープンソースですが、1つの例外を除いてはソフトウェア管理環境であるaptやyumで管理されているソフトウェアのみを導入しています。常用の環境ではソースコードをダウンロードしてコンパイルして利用するとか、バイナリを手動でダウンロードしてインストールするといったことはしていません。

ちなみに唯一の例外は、Ubuntuデスクトップ環境にインストールしているVirtualBoxです。これのみ提供ベンダから直接ダウンロードして利用しています。システム全体のソフトウェア管理システムの枠組みに入っていないような独立しているアプリケーションを使う場合、アプリケーション自身にアップデートの能力が必要です。VirtualBoxは、新しいアップデート版が用意されたとき、メッセージが表示されます。筆者はそのタイミングで、ブラウザから手動でダウンロードし、新しいバージョンをインストールします。

旧来のPCのシステムは、筆者がVirtualBoxで行っているような形でアプリケーション単位／ベンダ単位でアップデートをしていました。システムが一貫して管理するのではなく、アプリケーションを提供するベンダが独立して管理するわけですから、システムの中に複数のアップデート・ポリシー（含

むアップデート機構）が複雑にからみあいながら存在していることになります。最新のPC環境はその時代に比べると進化しており、たとえばWindows 10になってWindowsアプリストアの枠組みが用意されたので包括的な管理が可能になっています。

とはいえ、旧来の「入れたまま」のアプリケーションもまだまだ多く見受けられます。利用者登録をすることでセキュリティ・アップデートがメールなどで通知されるようであれば、まだ良いほうです。ベンダのWebサイトにいつの間にか告知が載っているだけという製品もいまだにあります。あとはユーザがWebサイトにアクセスし、ブラウザでダウンロードを行い、手順書を見ながら手動で設定ファイルなどのバックアップを行ったあとにアップデートしてくださいというわけです。みなさんも、そのような製品に心当たりがあると思います。

アップデート配布サーバから拡散したPetya

ウクライナで蔓延（まんえん）し、大きな被害を出したランサムウェア「Petya」に関しては、すでに前回（2017年9月号）で取り上げました。ウクライナのソフトウェア会社M.E.Doc社が提供する税務会計アプリケーション「MEDoc」は、税務当局へ提出するフォーマットをサポートしているため、ウクライナ国内では多くの企業に導入されています。MEDocのアップデート版は企業のアップデート配布用サーバから配布されるのですが、そのサーバはPetya騒動が起こる以前より、サーバに外部から侵入されている状態でした。そして、MEDocのアップデートのパッケージにPetyaが加えられ、配布用サーバの上にある本物と入れ替えられたのです。

コード・サイニングの警告は出るが

一般にユーザは、インストーラの形式で作られたアップデート版を取り込み、アップデート処理（やっていることはインストール処理）を行います。Windowsのインストール処理では、コード・サイニング[注1]をしていないものには警告が出ます。

日本国内では、正規に作っている商用アプリケーションのインストーラにコード・サイニングを行うのは常識です。コード・サイニングのために購入するデジタル証明書は有料であり、デジタル証明書発行サービスを提供している企業は、基本的には業務用途のためにサービスを行っているので、企業向けの所定の手続き（企業の存在を証明するものなど）が必要です。これらの手続きは日本や米国のような国ならば簡単でしょうが、ウクライナのような国でコード・サイニングの手続きにどのような手間がかかるかは、うかがい知ることはできません。

だたしこれも一般論ですが、現実的にコード・サイニングがどれだけ有効かということを知るのも難しいと思われます。日本でもそうですが、ハードウェア購入の際に付属してくるデバイスドライバなどは、コード・サイニングの警告が出るものが今でも多数あります。それを無視してインストールするのもよく見かける光景です。ですので、多くのユーザは「いつものことだ」と、いちいち気にかけていない可能性も十分にありえます。

ほかにも懸念があります。アプリケーションをシステムにインストールせず、アーカイブをファイルに展開し、ユーザのドキュメントフォルダなどに入れたままで利用するケースです。この場合、実行時には「信頼できるプログラムであるか」を確認する警告が表示されます。しかし、正規のサーバからダウンロードしてきているわけですから、ユーザは疑うことなく実行するでしょう。

それにアップデートの場合、ダイナミックライブラリであるDLLファイルを正規のものから不正なものに入れ替えられたら、実行開始時の警告は以前とまったく変わらないはずです。つまり、DLL入れ替えの場合はユーザが目視しただけでは違いを見つけることができない、ということになります。

アンチウイルスで防げるか

Petyaの事例のようにサプライチェーンを攻撃さ

注1） 開発元・配布元を明らかにし、内容が改ざんされていないことを証明するために、ソフトウェアやプログラムにデジタル署名をすること。

れて、そこ経由でシステムに侵入された場合、被害を防ぐには、最後の「スキャンしてマルウェアを発見するセキュリティツール」、いわゆるアンチウイルスツールしか頼れるものはないということになります。

マルウェアをスキャンして見つけ出すタイプの主流は、シグニチャの検出です。シグニチャとは過去にマルウェアとして発見された実行ファイル（この場合はおもにバイナリファイルですが、マルウェアが必ずしもバイナリファイルであるとは限りません）の中にある特徴的なパターンのことです。そのパターンをデータファイルとして用意し、スキャン対象となった実行ファイルの中に同じパターンを含んでいるかどうかを判定します。

この方法だと未知のものを発見するのは難しいということになります。また、新しいマルウェアのシグニチャを作成したとして、アップデートしたシグニチャを含んだデータファイルを配布し、それを利用できるようになってはじめて新しいマルウェアを見つけることができます。このようなしくみのため、どうしても新種発見から対応までタイムラグができてしまいます。

そこがPetyaの大きな問題点です。MEDocのアップデートファイルが用意できたとします。新しいアップデートができたことがなんらかの形で通知され、ユーザがアップデートファイルをダウンロードし始めます。MEDocのアップデートファイル経由でマルウェアを配布しようと計画をたてていた犯人が入れ替えを行います。そこから先はマルウェアが配布されるようになります。

まだ誰も知らない新規のアプリケーションを経由して広げるわけではなく、すでに広まっているアプリケーションを経由するわけですから、そのアップデートのペースも速いであろうことは想像がつきます。つまり、このマルウェアは入れ替えられた瞬間から、どんどんダウンロードされるのです。実際に、Petyaの感染は極めて速いペースで進んでいました。

このようにサプライチェーン攻撃は極めてやっかいなものであり、極めて効果的に広がるのです。

狙われたChromeのエクステンション

アプリケーションのインストールだけではなく、アプリケーションのプラグイン／エクステンションなどでも同様の問題が発生しました。

ブラウザ「Chrome」のエクステンションはChromeウェブストア経由で提供されています。もちろん、最新のソフトウェアのサプライチェーンのモデルを採用しています。Chromeウェブストア側でアップデートした場合、自動的にブラウザ側もアップデートするようになっています。その状況でユーザ数が100万人を越える有名なWeb開発用エクステンション「Web Developer 0.4.9」に不正なコードが加えられてしまうという事件が起こります[注2]（なお、2017年8月10日時点ではWeb Developer 0.5にアップデートされています）。作者のChris Pederick氏は8月3日の自らのツイートでフィッシングにひっかかりGoogleアカウントを乗っ取られてしまったと説明しています。

加えられた不正なコードとは、Webサイトに不正なJavaScriptが用意されていた場合、ブラウザに読み込んできてそれを自動実行してしまうという内容でした。報道によれば、そのJavaScriptは「PCを修理するよう促す偽の警告を表示したのち、アフィリエイトプログラムにリダイレクトする[注3]」といったものでした。

さらに、以下のエクステンションも同じように乗っ取られていたとのことです。

- Chrometana 1.1.3
- Infinity New Tab 3.12.3
- CopyFish 2.8.5
- Web Paint 1.2.1
- Social Fixer 20.1.1

注2）“Chrome web dev plugin with 1m+ users hijacked, crams ads into browsers”
https://www.theregister.co.uk/2017/08/02/chrome_web_developer_extension_hacked/
注3）“「Google Chrome」拡張機能が不正改変、開発者アカウント情報盗難--研究者報告” https://japan.zdnet.com/article/35105920/

これらのユーザ数を単純に足していくと480万ユーザとなるようです[注4]。

今後のサプライチェーン

2003年12月2日とかなり昔のことですが、Debianのサーバが侵入されたことがありました。このときは配布ファイルを改ざんされるようなことはありませんでした。現在、GNU/Linuxのディストリビューションで採用しているaptやyum、その他のパッケージシステムは、配布するソフトウェアにデジタル署名が行われており、途中で改ざんされたとしても検出できるようになっています。ダウンロードをする通信路を暗号化するだけでなく、ファイル本体にもデジタル署名をかけるなど、安全性を高めるしくみが複数入っています。

オープンソースやプロプライエタリなソフトウェアなどライセンスがいろいろありますが、このことをもってサポートや脆弱性対応が良いとか悪いとかの議論は成り立ちません。中身が公開されていないプロプライエタリなソフトウェアでもサポートが十分でないものや、信頼できるサプライチェーンを用意していないものもたくさんあるからです。

むしろ、GNU/Linuxのようにディストリビューションでサポートされているアプリケーションのみを利用したほうが、アップデート・ポリシーが一貫していて扱いやすいのではないかと筆者は考えるくらいです。これはオープンソースだからというわけではなく、サプライチェーンが明確化されているという利点です。ただし、そのように多くのアプリケーションをディストリビューションのパッケージという形で用意できるのはオープンソースだからという側面はあります。

現在のスマートフォンがそうであるように、今後はPCもストア経由でアプリケーションを導入したり、アップデートしたりするサプライチェーンの一本化の流れに収束していくでしょう。

警視庁はサプライチェーンを理解しているのか？

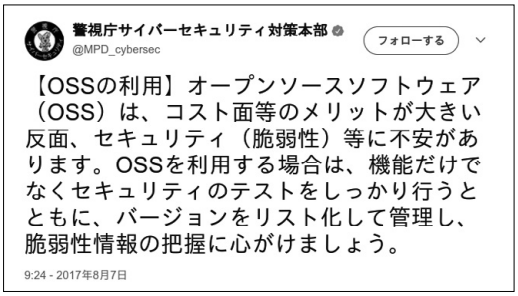
8月7日に警視庁サイバーセキュリティ対策本部（@MPD_cybersec）が**図1**のようなツイートをしました。

ここまでサプライチェーンの話をしてきたので、もうご理解いただけるでしょう。そのソフトウェアがセキュリティ面で安心できるかどうかを判断するには、オープンソースかプロプライエタリかといったことではなく、しっかりした信頼のおけるサプライチェーンを提供しているかどうかを見分けるのがポイントです。

警視庁は、機能だけではなくセキュリティのテストをしっかり行うことというアドバイスを述べていますが、たとえばOpenSSLやApacheといった大規模かつ複雑なオープンソースのセキュリティのテストをしっかりと行うことは筆者でもできません。App StoreやGoogle Playストアでも数多くのオープンソースのアプリが登録されています。ましてやLinuxカーネルなどどうするのでしょうか。警視庁のアドバイスはまったく非現実的であることがわかっていただけると思います。

ここで必要なのは、信頼があり安定しているサプライチェーンを提供しているソフトウェアを導入することです。だからこそサプライチェーンへの攻撃は重大な脅威であり、攻撃がいったん発生すると結果として大きな被害につながるのです。その点をしっかり理解しておかなくてはいけません。SD

◆**図1　警視庁サイバーセキュリティ対策本部のツイート**



注4）"PSA: 4.8 Million Affected by Chrome Extension Attacks Targeting Site Owners"
https://www.wordfence.com/blog/2017/08/chrome-browser-extension-attacks/

49

Debian Developer　やまねひでき　henrich@debian.org

Debian 10“Buster”に向けた取り組みが続々始動

Debian Hot Topics



Debian 9リリース直後のネタ枯れとイベント参加のために、2ヵ月ほどお休みをいただいておりました本連載、まだまだ続けるつもりでいます。読者のみなさんはご感想を、メール・Twitterでのツイート・編集部へのお便りなどでお寄せください(懇願)。

Debian 8.9/9.1リリース

7月22日にDebian 8.9および9.1がリリースされました。いつものとおりの不具合修正が中心で、目立った変更点はありません。

筆者が作業した部分では、大量のディスクが接続・認識されているシステムにおいて、/dev/sdaa以降のディスクへのブートローダーのインストールが失敗する問題の修正が入りました。そのようなシステムをお持ちの方はDebian 9.1をお試しください。

CPUの不具合が話題に

Debianに限った話ではないですが、Debianのメーリングリストに投稿された内容が話題となったのでご紹介を。「Skylake、Kaby Lakeと呼ばれる比較的最近のIntel CPUで、ハイパースレッディング機能を有効にするとランダムにデータが壊れる」という不具合についての報告がdebian-develメーリングリスト[注1]に投稿されました。対応としてはハイパースレッディングを無効にするか、ベンダが提供する修正版のBIOSやUEFIを適用すれば良いのですが、影響範囲がとても広いことに加え、多くのPCでは修正されたBIOSがベンダから提供されていなかったため、注目が集まったようです。

似たところで、AMDのCPU「Ryzen」の一部ロットには、Linux/*BSD上で非常に高いワークロードをかけ続けるとセグメンテーションフォールトをはじめとする問題が起こる現象があります。筆者の周辺でも話題になりました。

どこに不具合が潜んでいるかは明言されてはいないものの、AMD側でも問題が起こることは認識しており、対応を進めているとのことです[注2]。このような不具合に遭遇した場合はAMDカスタマーケア窓口に問い合わせると良いでしょう(多くの場合は、CPUの交換対応を行っていただけているようです)。

Liveイメージの廃止?

Debian 9“Stretch”で、Liveイメージの作成ツールが「live-build」から「live-wrapper」へ切り替えられました。しかし、コードがまだ成熟していないため、作成されるLiveイメージがいくつものバグを抱えてリリースされるという問題が起きました。正式リリース前にもテスト用のLiveイメージは継続的にビルドされていた

注1) URL http://deb.li/KkHM

注2) URL https://goo.gl/UqBKP4

ものの、誰からも指摘がなく、そのままバグに気づかずにスルーしてしまいました。そして、Debian 9リリース直後にいくつものバグ報告を受けて修正版の9.0.1を出す(その後もバグ修正に追われる)羽目になりました[注3]。

修正作業を行っているSteve McIntyreさんは、インストーライメージ作成全体の担当者ではあるものの「Liveイメージについてはこれまで(単なるユーザとしても)触る必要もなかったためにリリースまで詳しくチェックしていなかった」とのこと。心ないユーザからは嫌がらせの匿名メールが送られてきて、相当嫌になった様子で「Debianユーザが満足のいくLiveイメージをリリースするには、Liveイメージの開発プロセスに継続して一心に取り組んでくれる人がもっと必要だ。目に見える改善がなされない場合はLiveイメージのビルドを止めよう」[注4]と述べています。

一部では今回の件を取り上げて「DebianではLiveイメージがなくなるようだ！」などという記事が書かれるなどしていましたが、「担当者としてメンテする気力がない」という表明だけなので、積極的に廃止されるような事態にはならないと思われます[注5]。

/usr/bin/nodeがnodejsパッケージ提供に

上記の見出しだけを見ると「何を当たり前のことを？」と思われるかもしれません。じつはDebianではNode.jsが出る前から、/usr/bin/nodeは「ax25-node」というアマチュア無線ネットワーク用のパッケージが提供していました。そこで、ファイル名の衝突を避けるためにnodejsパッケージの実行ファイルは「/usr/bin/nodejs」として提供されていました。

この変更のため、DebianではNode.jsのパッケージマネージャーnpmで入れたモジュールが動作せず、/usr/bin/nodeにsymlinkを張ってくれるnodejs-legacyパッケージを別途インストールせねばなりません(Debianパッケージになっているnpmモジュールはいちいちshebangを/usr/bin/nodejsにしています)。

このようなしがらみによって、Debianでは中途半端に複雑な状況になっていたのですが、Debian 10“Buster”ではこれが改善されるようです。ax25-nodeパッケージがアーカイブから削除され[注6]、Debian 9“Stretch”以降では存在しなくなりました。そこで、技術委員会で再裁定が行われ、nodejsパッケージがそのまま/usr/bin/nodeを利用しても良いことになったのです。

これを受けてnodejsパッケージのバージョン6.11.2~dfsg-1から/usr/bin/nodeが提供されるようになりました[注7]。あとは、nodejsパッケージが依存しているlibv8(現在はlibv8-3.14.5パッケージ)が安定して使えるようになると良いのですが、ほぼ数時間ごとにリリースがされるupstream[注8]とのバージョンの乖離(かいり)を見る限り、はるか長い道のりのようです……。

opensslパッケージがTLS 1.0/1.1を無効に

openssl パッケージ1.1.0f-4での更新で唐突に、「TLS 1.0および1.1を無効にして、1.2を唯一のサポートSSL/TLSバージョンとする」という変更が入りました[注9]。事前調整と予告がなかったうえ、そのままでは確実に接続できない機器、サイト、クライアントが発生するという少々ラディカル過ぎる変更であることもあり、(予想どおり)多数の反発が出ています。一方、数は少ないですが賛成者もあり、Tollef Fog Heenさんは次のように述べています。

「自分としてはBusterではTLS 1.0が使える状態でリリースするべきだと思うが、開発サイクルの一部として無効にするのは非常に有用だ

注3) URL http://deb.li/ikPa1
注4) URL http://deb.li/394nl
注5) 対策についても一部で検討はされていますので、そのあたりは次号以降にご紹介できればと思います。
注6) URL https://bugs.debian.org/797929
注7) URL http://deb.li/sKz4
注8) URL https://github.com/v8/v8/releases
注9) URL http://deb.li/3wNvx

と思う。なぜならTLS 1.2を使う場合にパッケージにあるバグをあぶり出してくれるからだ。バグを見つけて修正するのはいいことだ」。

筆者も同様の意見で、さらに追加で次の理由を挙げます。

- 現状、**テスト版と不安定版だけに影響があり、多くのユーザが使う安定版には影響がない**
- 開発サイクルの最初期での変更で、リリースにはまだまだ多くの考える時間がある
- 悪い影響が多過ぎるようなら、Debian 10 "Buster"のリリース前に変更を切り戻すこともできる
- 変更を切り戻したくない場合、代替パッケージとして(通常使うべきではない)SSH1プロトコルに対応したopenssh-client-ssh1パッケージのような提供も可能だろう

変更時のchangelogにも「この変更はBusterリリース前に再考される」とも明記されています。そのため、最終的にはDebian 10 "Buster"で何の代替もなくTLS 1.0/1.1がまったく使えない状態にはならないものと推測していますが、今後の推移を見守りましょう。

Emacsが「emacs」パッケージのみの方向へ

これまでのemacsパッケージはEmacs 24・Emacs 25のようにバージョンがついた形で提供されてきました。しかし、Debian 10 "Buster"以降ではバージョン番号が付かない単一の「emacs」パッケージとして提供されるようです。開発がかなり昔に終わっているとても古いバージョンのEmacsが、ユーザ環境で残ってしまってEmacs関連パッケージについてバグ報告されるような事態が避けられるようになるとのこと。

また、これと並行してemacs lisp関連のパッケージの作成は「dh-elpa」の利用が推奨されるようになってきています。ELPA(Emacs Lisp Package Archive)[注10]はEmacsのパッケージリポジトリで、このELPAから導入したEmacs Lispパッケージと互換性を持たせる形でDebian側のEmacs Lispパッケージを作成するのがdh-elpaです。

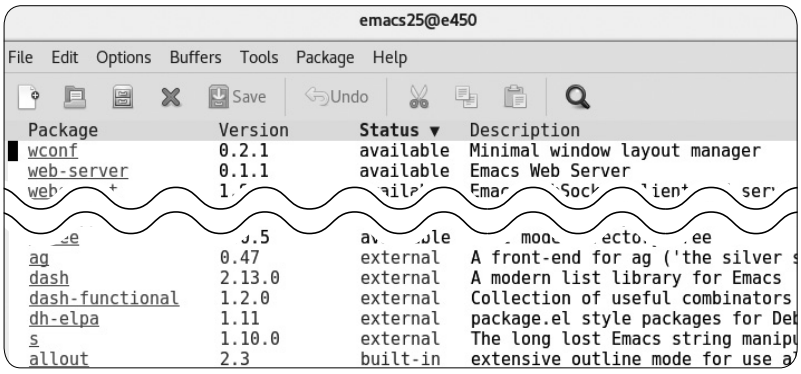
ユーザが導入できるパッケージとシステム側のパッケージの整合性が取れるという理想的な形になるのに加え、仕様が変更された場合でもdh-elpaを更新後に関連パッケージをリビルドするだけで済むので作業量が少なくなるという利点があります(これはdebhelperを使うのと同様の利点です)。まだまだ開発が始まったばかりではありますが、dh-elpaを使ったパッケージは「elpa-xxx」という名前で提供されています。現在、unstableでは160個ほど提供されていますので、興味のある方は確認してみてください(**図1**)。

Debianポリシードキュメントをモダンな形に

Debianパッケージは「Debianポリシー」と呼ばれるドキュメントに沿ってその造りが決まっており、これをもとに「Lintian」というツールでポ

注10) URL http://elpa.gnu.org/

▼図1　elpa-agを入れた場合の例。Emacsのパッケージ管理(M-x list-packages)でもStatusが「external」となって参照できる



リシーに準拠しているかどうかの妥当性を確認しています(パッケージの仕様書を定めてそのとおりになっているかテストを回している、とご理解ください)。このサイクルが回ることで、品質の高いパッケージの作成と維持が可能になっています[注11]。そのDebianポリシーの記述に、現在はDocBook XMLを使っていますが[注12]、これを本誌でも連載されていた「Sphinx」[注13]と「reStructuredText(reST)」[注14]を使う形にしよう!という提案を筆者が行いました。

DocBook XMLは、「OSSの世界では一般的な文書形式であり、枯れて安定したツールチェイン群に支えられている」「非常に細やかなセマンティクスの付加が可能」という利点があるものの、「可読性が低く編集がしづらい」という弱点があります。このため、メンテナの負荷が高く新規参入者が入ってきづらい(バグ報告で意見は言ったとしても、パッチの形まで落とし込んでの提案がされにくい)という状況にありました。

これを脱却すべく「もっと可読性の高い、編集が容易な形式に変えよう」ということでreSTを採用し、ツールとしてSphinxを使ってはどうか?と問いかけたのです[注15]。

提案を行ったメーリングリストでは、「先日ようやくDebian SGMLからDocBookへのコンバートが終わったというのに、ほかの形式にしろというの?」という疑問の声も出ましたが、「いや、DocBookにしないとPandoc[注16]がreST形式に変換できないから、その作業によってSphinxへの道が開かれたんだよ」というフォローがすぐに入ります。「リグレッションがあるかもしれないけど、可読性が高まるのは『相当なもの』だね」というコメントがポリシーチームのRuss Allberyさんから寄せられるなど評価は上々でした。すでにLinuxカーネルのドキュメントが同じDocBookからSphinxへの移行作業を進めており[注17]、この点も提案の妥当性を後押ししてくれました。

最終的にDebConf17[注18]で、ポリシーチームのRuss AllberyさんとSean Whittonさん(**写真1**)と筆者で簡単なSprint[注19]を実施し、Sphinxを使った形に変更することがバージョン4.1.0でできました。

さらに次のステップとしては、過去のDebian SGML形式ではマージが難しかったDebian JP Projectで保守していた日本語訳[注20]の本体への追加を見込んでいます。

今回の提案によってDebianポリシー自体の改善が進み、(とくに日本語版など多言語での提供で)より品質の高いパッケージを作るために必要なポリシーを読み込む負荷も下がることが期待されます。SD

注11) あくまでもパッケージの品質のことであり、ソフトウェア自体の品質とは別です。パッケージの品質は筆者の観点からは「メタデータの品質」であるように思います。
注12) じつは、Debianポリシーのドキュメントは先日まで可搬性のない、さらに原始的な「Debian SGML」という独自のSGML形式でした。ここからようやく脱却して一般的に使われているDocBook形式にコンバートされたのが、つい先日の2017年6月のことです。
注13) Python製のドキュメントビルダー。本誌2015年4月号～2017年3月号の連載記事「Sphinxで始めるドキュメント作成術」を参照。
注14) マークアップ言語の1つ。昨今はMarkdownやAsciidocなどと並んでよく利用されています。
注15) Debianポリシーのためにわざわざ DocBookをこれから習得しよう!などという酔狂な人はなかなか現れないだろう、という思いもありました。reSTは利用者が多く習得するのも容易なマークアップ言語なので、パッチを送ってくる人も増えるだろう、という目算です。
注16) URL http://pandoc.org/ Haskellで書かれた文書形式の変換ツールで、非常に多彩な形式での入出力が可能(とはいえ完璧ではないので、手直しが必要になります)。
注17) URL https://youtu.be/jY_C-Z0qMSo
注18) Debianカンファレンス。2017年はカナダ・モントリオールで8月6日～12日に開催。次回で報告予定です。
注19) 短距離走にたとえ、短期間で集中して作業するイベントのことを指します。
注20) URL http://deb.li/cGRg 更新が数年前で止まっているのでブラッシュアップが必要で、作業者を随時募集しています。

▼写真1 DebianポリシーチームのRuss Allberyさん(左)とSean Whittonさん(右)



OpenVPNでおうちネットワークにログイン

Ubuntu Japanese Team
あわしろいくや

今回はOpenVPNサーバを動作させて自宅内のネットワークにログインする方法を紹介します。

OpenVPNの今

ちょっと前まで、手軽なVPNの実装はPPTPでした。比較的安価な家庭用のルータでもPPTPサーバ機能があるものが販売されています。またWindowsやmacOSはもちろんモバイルデバイスにもPPTPクライアント機能があり、手軽に使用できていました。

しかしPPTPは脆弱であることが以前より判明しており、iOSやmacOSの最新版ではPPTPクライアント機能が削除されています。たしかに手軽とはいえセキュリティの強固さのほうが優先されるべきであり、PPTPの使用は推奨されません。

昨今はASUS、NETGEAR、TP-Link、SynologyなどがOpenVPNに対応したルータを販売しています。はたまたOpenVPN専用のルータなるものまで発売されています[注1]。ファームウェアのアップデートなどさまざまなことを自動的にやってくれるのはいいのですが、同時接続数に5台までという制限があるのが気になりました。

このようにOpenVPNは確実に普及してきていますが、問題はOpenVPN自身のセキュリティです。OpeVPNはセキュリティ全般をOpenSSLに依存しており、すなわちOpenSSLの安全性がOpenVPNの安全性と言えるわけです[注2]。

OpenSSLの脆弱性については記憶に新しいので繰り返しませんが、Ubuntu 16.04 LTSのリリース後から現在に至るまでOpenSSLは8回のアップデートがあり、そのうち4回が脆弱性の修正です。

Ubuntuだとパッケージのアップデートをすれば脆弱性への対処が完了しますが、ルータの機能となるとファームウェアのアップデートを待たなくてはならず、どうしても即座に修正とはいかなくなります。しかもそれだけの高頻度ともなると不安が拭えません。というわけで、ルータの機能に頼るのではなく、UbuntuにOpenVPNサーバを構築し、外出先から自宅のネットワークに接続するといいのではないかという結論に達しました。

Ubuntuで動かすのであればOpenVPNにこだわる必要もないのですが、サーバの設置が簡単なこと、Network ManagerのOpenVPNプラグインが簡単に使えること、WindowsやモバイルデバイスにおいてもOpenVPNクライアントをインストールすれば簡単に使用できることなどから、OpenVPNがベストなのではないでしょうか。

仮想マシンの設定

OpenVPNサーバを動かすUbuntuはどこにインストールするのかというのは悩ましく、実機だったりコンテナの中だったりいろいろあると思うのですが、今回はVirtualBoxにしました。パフォーマンスを求めないのであれば、Raspberry Piシリーズなどのシ

注1） https://www.planex.co.jp/products/vr500-ov1/

注2） これはOpenVPN自身に脆弱性がないことを意味するわけではありません。現にこの6月にOpenVPNの脆弱性が修正されています。

ングルボードコンピューターでもいいでしょう。

VirtualBoxの設定のコツとしては、「ネットワーク」で「割り当て」を「ブリッジアダプター」にすることと、「プロミスキャスモード」を「すべて許可」にすることです（**図1**）。前者は「NAT」にしてポートフォワードの設定をしてもいいはずです。

ルータの設定

OpenVPNはUDPのポート1194を使用するので、これをOpenVPNがインストールされているUbuntuに対してポートフォワードします。ついでにOpenSSHのポートも通しておくと、何かあったときに確認できて便利です。

OpenVPNサーバの設定

Ubuntu（通常はUbuntu Serverでしょうが、デスクトップでも問題はありません）をインストール後、VirtualBoxの「ネットワーク」にある「割り当て」を「ブリッジアダプター」にした場合は固定IPアドレスに変更します。事前準備はそのくらいです。

OpenVPNサーバの設定方法はおおむねUbuntuサーバガイドに書かれているとおり[注3]ではありますが、一部変更が必要なところがあります。

まずは必要なパッケージをインストールします。

```
$ sudo apt install openvpn easy-rsa
```

続いてPublic Key Infrastructure（PKI）の設定を行います。

```
$ sudo mkdir /etc/openvpn/easy-rsa
$ sudo cp -r /usr/share/easy-rsa/* /etc/openvpn/easy-rsa/
$ sudo editor /etc/openvpn/easy-rsa/vars
```

/etc/openvpn/easy-rsa/varsは**リスト1**の部分を編集します。これもサンプルですので、は適宜編集してください。もちろん内容は架空のものであり、`"Ikuya Company"`は実在しません。

ここからはsudoで実行するのが難しいので、**sudo -s**コマンドでルート権限になってから実行します。

```
$ sudo -s
# cd /etc/openvpn/easy-rsa/keys/
# source vars
# ./clean-all
# ./build-ca
（すべてEnterキーを押す）
```

下準備が終わったので、サーバ証明書を作成します。

```
# cd /etc/openvpn/easy-rsa/
# source vars
# ./build-key-server（サーバ名）
（キー操作を行う）
# ./build-dh
# cd keys/
# cp（サーバ名）.crt（サーバ名）.key ca.crt dh2048.pem /etc/openvpn/
```

続けてクライアント証明書を作成します。

```
# cd ../
# source vars
# ./build-key（クライアント名）
# exit
```

なお今回はサーバ名をopenvpn、クライアント名をclientとしましたが、もっと区別のしやすいものにしたほうがいいでしょう。

注3） https://help.ubuntu.com/lts/serverguide/openvpn.html

図1　VirtualBoxのネットワークの設定例



リスト1　/etc/openvpn/easy-rsa/vars

```
# These are the default values for fields
# which will be placed in the certificate.
# Don't leave any of these fields blank.
export KEY_COUNTRY="JP"
export KEY_PROVINCE="Osaka"
export KEY_CITY="Osaka"
export KEY_ORG="Ikuya Company"
export KEY_EMAIL="ikunya@gmail.com"
export KEY_OU="OpenVPN"
export KEY_CN="VPNTest"
export KEY_ALTNAMES="OpenVPN"
```

クライアントからの接続に必要なファイルは、

- /etc/openvpn/ca.crt
- /etc/openvpn/easy-rsa/keys/(クライアント名).crt
- /etc/openvpn/easy-rsa/keys/(クライアント名).key

の3つです。これらを安全な方法でクライアントにコピーします。最も手軽で安全な方法は、scpコマンドでコピーすることでしょう。

いよいよサーバの設定に移ります。サンプルファイルをコピーして、それを編集します(**図2**)。とはいえ変更するところはあまり多くありません。**リスト2**を作成した証明書と秘密鍵のファイル名に合わせるくらいです。

OpenVPNサーバの設定はこれで終わり、カーネルの設定を変更します。**/etc/sysctl.conf**の**#net.ipv4.ip_forward=1**のコメントを外し、**net.ipv4.ip_forward=1**としてから次のコマンドを入力してください。

```
$ sudo sysctl -p /etc/sysctl.conf
```

最後にOpenVPNサーバを再起動します。

```
$ sudo systemctl restart openvpn@server.service
```

必要最低限の設定はこれでおしまいですが、これだとVPNは張れても接続先のローカルネットワークにはアクセスできません。ふたたび**/etc/openvpn/server.conf**を編集します。140行目前後に、

```
push "route 192.168.3.0 255.255.255.0"
```

を追加します。"192.168.3.0"はお使いのネットワークのセグメントに合わせて変更してください。

200行目前後の、

```
client-to-client
```

のコメントアウトを外しておくといいでしょう。説明のとおり、クライアント同士でも通信できるようになります。

ほかにも設定ファイルを見てみて必要そうな設定があったら変更してみてください。設定が終わったら、やはり次のコマンドでOpenVPNサーバを再起動します。

```
$ sudo systemctl restart openvpn@server.service
```

OpenVPNクライアントの設定

OpenVPNクライアントはOpenVPNサーバとは違うUbuntuを用意します。自明とは思いますがお断りしておきます。

Ubuntuでは前述のとおりNetwork ManagerのOpenVPNプラグインを使用します。まずは必要なパッケージをインストールします。

```
$ sudo apt install network-manager-openvpn-gnome
```

右上のネットワークアイコンをクリックし、「情報を編集する」をクリックします。「Add」をクリックすると「接続の種類を選んでください」の中に「OpenVPN」があるはずですので、これを選択して「作成」をクリックします。もしない場合はインストールのし忘れが考えられます。

接続の詳細な設定に関しては**図3**と**図4**をご覧いただきたいのですが、ポイントは「ゲートウェイ」にOpenVPNサーバのホストあるいはIPアドレスを入力することと、「タイプ」を「証明書(TLS)」にして3つのファイルを読み込ませることです。あとは「詳細」をクリックして表示される詳細オプションの2つ

リスト2　サンプルファイルの変更箇所

```
# OpenVPN can also use a PKCS #12 formatted key file
# (see "pkcs12" directive in man page).
ca ca.crt
cert (サーバー名).crt
key (サーバー名).key  # This file should be kept secret
```

図2　サンプルファイルをコピーして編集する

```
$ sudo cp /usr/share/doc/openvpn/examples/sample-config-files/server.conf.gz /etc/openvpn/
$ sudo gzip -d /etc/openvpn/server.conf.gz
```

の設定は重要で、「LZOデータ圧縮を利用する」にチェックを入れないと接続できませんでした。「カスタムのトンネルMaximum Transmission Unit (MTU)を利用する」のMTUの値はサンプルです。これは設定しないと接続はできるもののファイルのコピーやWebサイトの閲覧に支障が出ます。

ここまで設定できたら「保存」し、右上のネットワークアイコンをクリックして「VPN接続」-「(接続名)」で接続します。

接続できたら、ファイルマネージャ(Files/Nautilus)を起動し、16.04だと「サーバーへ接続」をクリックし、「サーバーアドレス」に「smb://(ファイルサーバのIPアドレス)」などと入力すればファイルサーバにアクセスできるはずです。17.04以降では「他の場所」をクリックし、「サーバーへ接続」の横に前述の書式で入力します。

ファイルサーバがない場合は端末を起動し、ルータに対してpingを打ってみるといいでしょう。

デフォルトルートをOpenVPNのネットワークではなくインターネット接続のネットワークに変更したい場合は、OpenVPNクライアントの設定をふたたび開き、「IPv4設定」タブの「ルート」をクリックし、**図5**を参考にしてルートを変更します[注4]。「アドレス」をお使いのネットワークのセグメントに合わせればいいはずです。

応用例

VPSで何かのサービスを動かし、それをVPN経由で使用したいという場合には、VPSでOpenVPNサーバを動かし、自宅にOpenVPNクライアント機能つきルータを置けば手軽でいいのではないかと思い至りました。ASUSのルータの多くにはOpenVPNクライアント機能が付いており、ドキュメント[注5]によると.ovpnが必須のようですが、.ovpnファイルを生成するのはわりと簡単で、`<ca></ca><cert></cert><key></key>`のタグ中に3つのファイルの中身をコピー&ペーストするだけです。

なお、今回の例ではNATの中でサーバを動作させたのでファイアウォール関連の設定はまったく行っていませんが、VPSで動かす場合は必ずこの設定を行ってください。SD

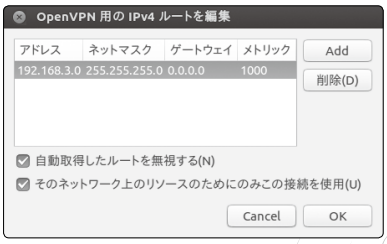
図3　OpenVPNクライアントの設定



図4　OpenVPNクライアントの詳細設定



図5　ルートの設定サンプル



注4) この方法は第79回(本誌2016年11月号)でも紹介したのですが、ルートの設定をいったん閉じたあとふたたび開くと誤った値が表示される(設定自体は正しく適用されている)というバグがあることに気づかず、誤った状態のスクリーンショットが掲載されてしまったことに最近気づきました。この場を借りてお詫び申し上げます。

注5) https://www.asus.com/jp/support/FAQ/1011232

SOURCES

レッドハット系ソフトウェア最新解説

第13回

OpenShiftの新機能 Ansible Service Broker

2017年8月上旬にOpenShift Container Platform 3.6がリリースされ、Ansibleを活用したアプリケーション管理機能が新しく導入されました。今回は簡単な利用方法を紹介します。

Author 小島 啓史(こじまひろふみ)
mail : hkojima@redhat.com
レッドハット㈱ テクニカルセールス本部 ソリューションアーキテクト

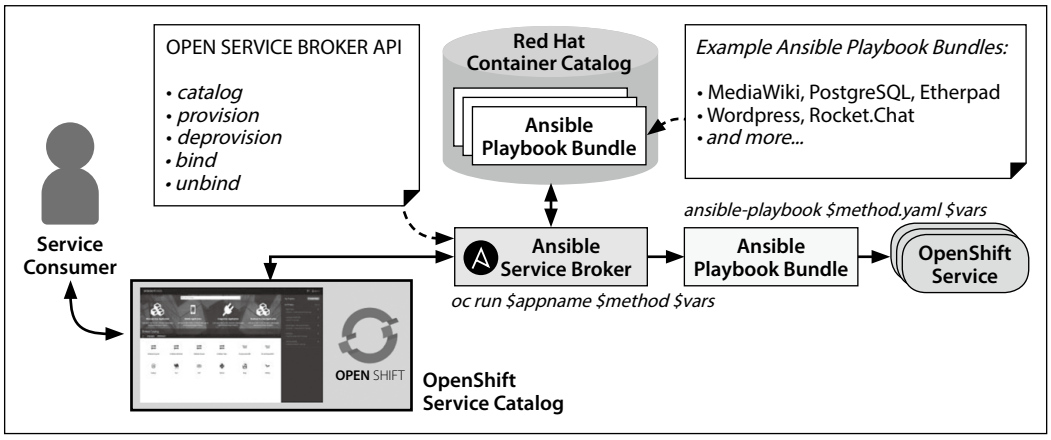
Ansible Service Brokerの概要

Ansible Service Broker[注1](以下、ASB)は、KubernetesやOpenShift環境上のリソースを管理する機能の1つであり、OpenShift Container Platform(以下、OCP)3.6からテクノロジープレビューとして新しく導入されました。これはCloud Foundryで外部サービスとの連携を行うために用意されたService Brokerという機能をもとにしており、Service Brokerが備えているAPIをオープン化[注2]したOpen Service Broker API[注3]を実装したものとなります。そしてAnsibleという名前が付いているとおり、リソースの作成／削除／連携／連携解除にAnsibleを活用しているのが特徴であり、こうした内容を記載したPlaybookの集合をAnsible Playbook Bundle[注4](以下、APB)と定義しています。ASBを活用することでKubernetesやOpenShift環境上のアプリケーションだけでなく、パブリッククラウドなどで動作するアプリケーションの作

注1) URL https://github.com/openshift/ansible-service-broker

注2) URL https://www.cloudfoundry.org/open-service-broker-api-launches-as-industry-standard/
注3) URL https://openservicebrokerapi.org/
注4) URL https://github.com/fusor/ansible-playbook-bundle

▼図1 Ansible Service Brokerのアーキテクチャイメージ



成や相互連携などをAnsibleで簡単に自動化できるようになりますので、アプリケーションの疎結合化の実現につながります。

ASBのアーキテクチャイメージは**図1**のようになります。ASBはOpenShiftにあるService Catalog[注5]経由で利用されます。カタログ参照やリソース管理などの動作はOpen Service Broker APIの仕様として定義[注6]されており、ASBはDockerイメージに含まれたAPBの中からユーザのリクエスト(アプリケーションの作成など)に対応したPlaybookを実行します。**図1**のアーキテクチャイメージで参照されているDockerイメージリポジトリは、Red Hat関連のソフトウェアや認定ISVのDockeイメージを格納したRed Hat Container Catalog[注7]となりますが、プライベートのDockerイメージのリポジトリを参照させることもできます。

Ansible Service Brokerの使い方

テクノロジープレビューとして新しく導入された機能ですが、簡単に試せるようになっています。早速、使い方を見てみましょう。まずはOCP 3.6をインストールする最新のRHEL7サーバを1台用意します。そしてOCP 3.6をインストールするための製品リポジトリを有効化します(**図2**)。

続いてOCP 3.6のインストールに利用するツールを用意します。インストール専用のAnsible Playbookがありますので、必要なインベントリファイルをダウンロードしてPlaybookを実行します(**図3**)。このあたりの詳細は、本誌2017年1月号の連載記事やOCPのAdvanced Installationページ[注8]を参照ください。

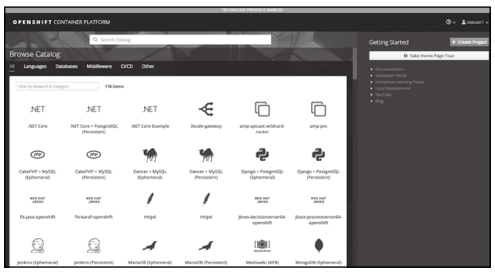
▼図2　RHELの登録と製品リポジトリの有効化

```
# subscription-manager register --auto-attach
# subscription-manager repos --disable="*"
# subscription-manager repos \
    --enable=rhel-7-server-rpms \
    --enable=rhel-7-server-ose-3.6-rpms \
    --enable=rhel-7-server-extras-rpms \
    --enable=rhel-7-fast-datapath-rpms
```

▼図3　OCP 3.6のオール・イン・ワン構成の作成

```
# yum -y install atomic-openshift-utils httpd-tools
# ssh-keygen -f /root/.ssh/id_rsa -N ''
# ssh-copy-id root@OCP_HOST_NAME
# htpasswd -c /root/htpasswd TEST_USER_NAME
# ansible-playbook -i sample-single-hosts  注9\
    /usr/share/ansible/openshift-ansible/playbooks
/byo/config.yml
```

▼図4　OCP 3.6のService Catalog画面イメージ



10～20分ほど待つとインストールが完了します。そのあと、

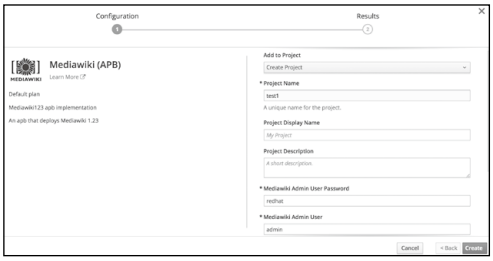
```
https://OCP_HOST_NAME:8443
```

にWebブラウザでアクセスして、htpasswdコマンドで作成したユーザ認証情報を利用してログインできることを確認します。ログインするとOCP 3.6のService Catalog画面が表示されます(**図4**)。Webコンソールに表示されたService CatalogからAPB関連のカタログを利用してみます。

OCP 3.6のデフォルトで用意されているAPB関連のカタログは、MediaWiki[注10]とPostgre

注5) URL https://docs.openshift.com/container-platform/3.6/architecture/service_catalog/index.html
注6) URL https://github.com/openservicebrokerapi/servicebroker/blob/master/spec.md
注7) URL http://jp-redhat.com/openeye_online/column/oss-note/5894/
注8) URL https://docs.openshift.com/container-platform/latest/install_config/install/advanced_install.html
注9) URL https://raw.githubusercontent.com/h-kojima/openshift/master/ocp3u6/ansible/sample-single-hosts
注10) URL https://github.com/fusor/apb-examples/tree/master/mediawiki123-apb

▼図5　APB関連カタログの利用イメージ



SQL[注11]の2種類です。これらを利用するには基本的にカタログから選択して、管理者パスワードなどをパラメータとして入力していけばいい(**図5**)のですが、MediaWikiコンテナの実行には執筆時点ではroot権限が必要になるので、次のコマンドを実行してroot権限が必要なコンテナの実行権限を各ユーザに与えておきます。また、このカタログで定義されているMediaWikiとPostgreSQLアプリケーションは、ともに永続ストレージ[注12]を必要とするので適宜設定を行う必要があります。

```
# oc adm policy add-scc-to-group anyuid
system:authenticated
```

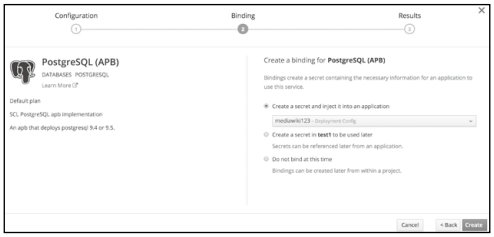
APB関連カタログからアプリケーションを新規作成すると、openshift-ansible-service-brokerという名前のプロジェクト(OpenShiftの名前空間)上のASBプロセスを起動しているPod(以下、ASBのPod)が、カタログを参照してAPB

注11) URL https://github.com/fusor/apb-examples/tree/master/rhscl-postgresql-apb
注12) URL https://docs.openshift.com/container-platform/3.6/architecture/additional_concepts/storage.html

▼図6　ASBプロセスを実行しているPodの情報参照イメージ

```
# oc project openshift-ansible-service-broker
# oc get pods
NAME
asb-1001951353-bsrjk
# oc logs -f asb-1001951353-bsrjk
  ……(略)……
[NOTICE] Ansible Service Broker Started
[NOTICE] Listening on http://0.0.0.0:1338
[INFO] AnsibleBroker::Catalog
  ……(略)……
[NOTICE] Creating pod "apb-2e145a9e-9f9a-4993-ae7e
-c1f796e2e253" in the test1 namespace
```

▼図7　既存アプリへのPostgreSQL認証情報の紐付けイメージ



を実行するための専用コンテナを一時的に起動してアプリケーションを作成していることを、「oc logs」コマンドでログを見ることで確認できます(**図6**)。

またPostgreSQLアプリケーションを作成するときには、MediaWikiアプリケーションに接続するための設定ができます(**図7**)。この場合はPostgreSQLアプリケーション作成時に設定されるDB認証情報のOpenShiftのSecretリソース[注13]としての登録と、MediaWikiアプリケーションへの紐付けをASBのPodが自動的に実行します。こうした認証情報の紐付けについては、MediaWikiアプリケーションを起動しているPodの情報を「oc export pods」コマンドなどで見ることで確認できます。なお、紐付けしたあとはMediaWikiアプリケーションを実行しているPodの設定が変更されているので、WebコンソールかCLIを利用してPodを再デプロイする必要があります。

◆　◆　◆

今回紹介したMediaWikiとPostgreSQLアプリケーションを作成するためのデフォルトカタログのほかにも、APB作成ツールを利用すればカスタムアプリケーションのカタログも作成できます。今後のOCPではこうしたツールも含めてパブリッククラウドとのサービス連携機能も追加・強化していく予定です。興味がありましたら、ぜひ試してみてください。SD

注13) URL https://docs.openshift.com/container-platform/latest/dev_guide/secrets.html



第66回

RACKによるTCPのパケットロス検出

Text：青田 直大　AOTA Naohiro

8月20日にLinux 4.13-rc6までリリースされました。だいぶ、bug fixも落ち着いているようで、27日にはLinux 4.13がリリースされていることでしょう。この記事をご覧いただくころにはとっくにLinux 4.13がリリースされ、きっとみなさんは楽しく使っているのではないかと思います。

今回は従来のTCPのパケットロス検出方法を根本から置き換えるRACKについて紹介します。

TCPのパケットロス検出

TCPは、そのパケットのデータの到着と順序を保障するプロトコルです。この保障を実現するため、TCPのパケットにはシーケンス番号が付いており、この番号を見ることで通信経路上での順序変更や欠落を受信側は検知できます。

パケットロス検出のアルゴリズムはいくつかありますが、最も基本的なアルゴリズムであるNewRenoの動きを簡単におさらいしておきましょう。前述したように、TCPではシーケンス番号を付けてパケットを送信します。パケットを受け取った側は「次に受信を期待する」シーケンス番号を付けて受信確認のACKを返します。たとえば、**図1左**のようにシーケンス番号が100でパケットのデータ長が100の場合、ACK = 200が返されます。

ここで送信側が続けてシーケンス番号がそれぞれ200、300、400、……のパケット(データ長はいずれも100)を送るとします。ここでシーケンス番号200のパケット(以下P200のように表記)が失われたとしましょう。受信側には、P200を期待していたのに、異なるシーケンス番号のパケットであるP300が到着しています。このとき、受信側は期待するシーケンス番号であるACK=200を再度返します。このACKが送信側に届くと、送信側はこれを「重複したACK」(DupACK)と認識します。さらにP400、P500が到着すると、受信側は同様にACK = 200を返していきます。この両ACKが到着すると、3つのDupACKを受け取ったことになります。このとき、送信側はP200が失われたと判断し、P200を再送します(Fast Retransmit)。

ここで、「重複したACK」を受信したらすぐにパケットロスとみなさず、3つのDupACKを待つのは、パケットの順序入れ替えへの耐性を持つためです。P100、P300、P200とパケットが入れ変わって到着した場合でも、P300に対してACK = 200のDupACKが返ります。1つ、2つのDupACKでパケットロスとみなしてしまうと、1つまたは2つの順序入れ替えもパケットロ

スとみなして無駄にパケットを再送してしまうことになります。

また場合によっては、このDupACKによるロス検出ができないことがあります。たとえば、P100からP400と4つのパケットだけを送るケースを考えてみましょう。ここでP200が失われるとどうなるでしょうか。P300、P400はDupACKを発生させますが、この2つではパケットロス判定の基準に満たずP200が失われたとは考えません。こうした場合に対応するため、RetransmissionTimeout(RTO)というものが使われます。これは簡単に言えば、データを送ってから一定時間、新しいACKを受け取らなかった場合にパケットが失われたとみなす方法です。

このタイムアウトが短か過ぎると、ちょっとしたネットワークの遅延でRTOが発生し、パケットが再送されてしまいます。これではネットワーク帯域を無駄使いしてしまい、好ましくありません。ですので、RTOはおおよそRound-TripTime(RTT)(の推測値)の移動平均、または1秒のどちらか大きいほうと設定されます。こうなると、通信に大きな遅延が発生することになります。

このように、TCPは(1)DupACKを数える、(2)タイムアウトを待つの2つの方法でパケットロスを検出してきました。説明したように、タイムアウトの方法では通信を停滞させ、スループットを悪化させます。そこで、できるだけタイムアウトにたよらずパケットロスを検出できるようにさまざまな方法が提案されてきました。

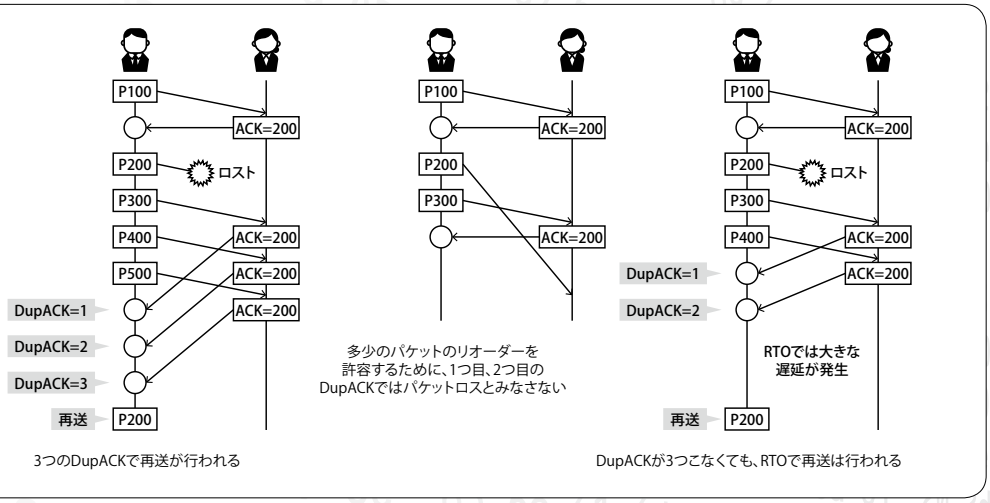
これらの方法は大きく2つに分けることができます。1つはSACKというTCPオプションを活用する方法で、もう1つは特定の状況下でDupACKをロスとみなす閾値(しきいち)を変更するものです。

選択的ACK：SACK

まず、第一の方法の前提となるSACKについて解説します。SACKはTCPのオプション機能の1つです。たとえば、送信側がそれぞれサイズ100でP100からパケットを送り、そのうちP200、P300と2つのパケットが失われたとします。受信側はP100に対してACK = 200を、そしてP400、500、600に対してACK = 200を返します。ここで送信側はP200が失われたと判定します。この時点で、送信側にわかるのは、P200が失われたということだけで、P300以降がどのような状態であるかはわかりません。このケースではP300からP600のうちいずれか3つが届いてDupACKが返っていることまでは推測できます[注1]が、どのパケットが届いてどのパケットが届いてい

注1) 実際には、ACKパケットが通信経路上で複製されるケースもあるので、こうとも言いきれません。

▼図1　NewRenoのパケットロス検出





ないのかはわかりません。確実に次のロスを検出するにはACKが返ってくるのを待つしかありませんが、これはやはり通信を停滞させます。ひとまずパケットが失われたと想定して、すべて送りなおすこともできますが、これはネットワーク帯域の無駄使いにつながります(**図2左**)。

そこで、ACKに加えてどの範囲のパケットを現在受信しているのかを送信側に伝えるのがSACKの機能です。先ほどの例で見ていきましょう。P100の受信の時点では、欠けたパケットはないので従来と同じくACK = 200を返します。その後P400に対してACK = 200とDupACKを返すのですが、この返答にSACKの情報として「400-500を受信済み」と一緒に載せます。同様に残りの2つのDupACKからはSACKの情報として「400-600を受信」、「400-700を受信」をそれぞれ送信側は知ることができます(**図2右**)。

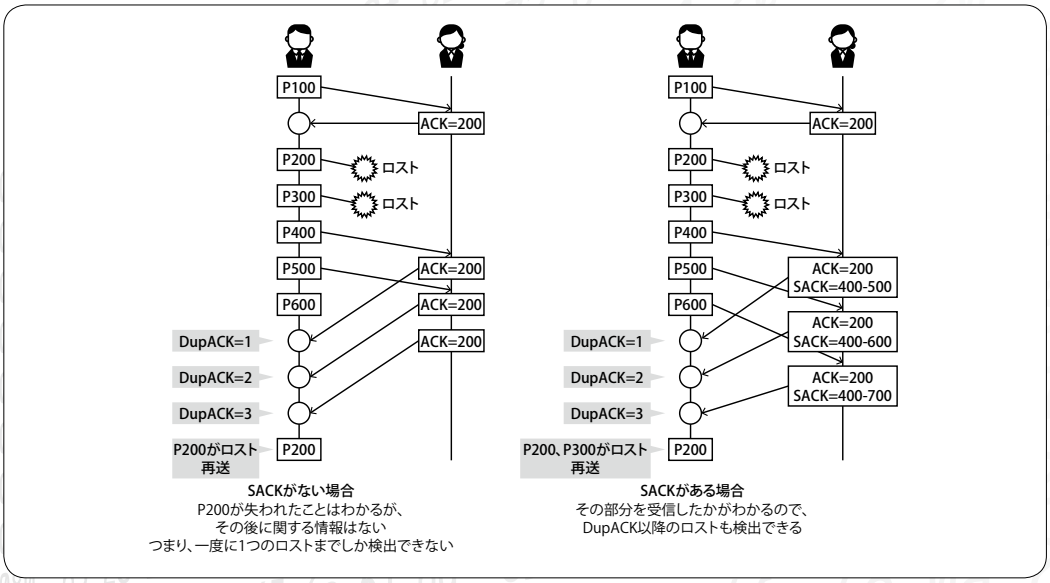


RFC 6675：SACKを用いたロス検出

こうしたSACKの情報をもとに、より効率的に失われたパケットを再送するアルゴリズムが提案されています。たとえば、RFC 6675で規定されたアルゴリズムがあります。この方法では、まだACKされていないそれぞれのパケットに対して、そのパケット以後のパケットが、一定の閾値(DupThresh)以上SACKされていれば失われたとみなします。DupThreshは、元のアルゴリズムでいくつのDupACKでパケットが失われたかと判別するときの閾値のことで、DupThresh = 3になっています。

具体例を見てみましょう。**図3**のように送信側はこれまで同様にP100からサイズ100のパケットを送っていき、そのうちP200、P300、P600が失われたとします。このとき、P700の到着に対して受信側は"ACK = 200、SACK = 400-600、700-800"と最初に失われたP200と、これまでに受信できている範囲を返します。P200に対しては、これが3つ目のDupACKですので失われたと(もともとのアルゴリズムで)判別できます。問題は残りのパケットです。P300はP400、500、700とP300より後のパケットが3つ(= DupThresh)分SACKされているので、失われたと判別されます。一方で、P600はまだP700しかSACKされていないため、リオーダーの可能性も高く、まだ失われたとは判別されません。

▼図2　SACKにより、到達したパケットを知ることができる

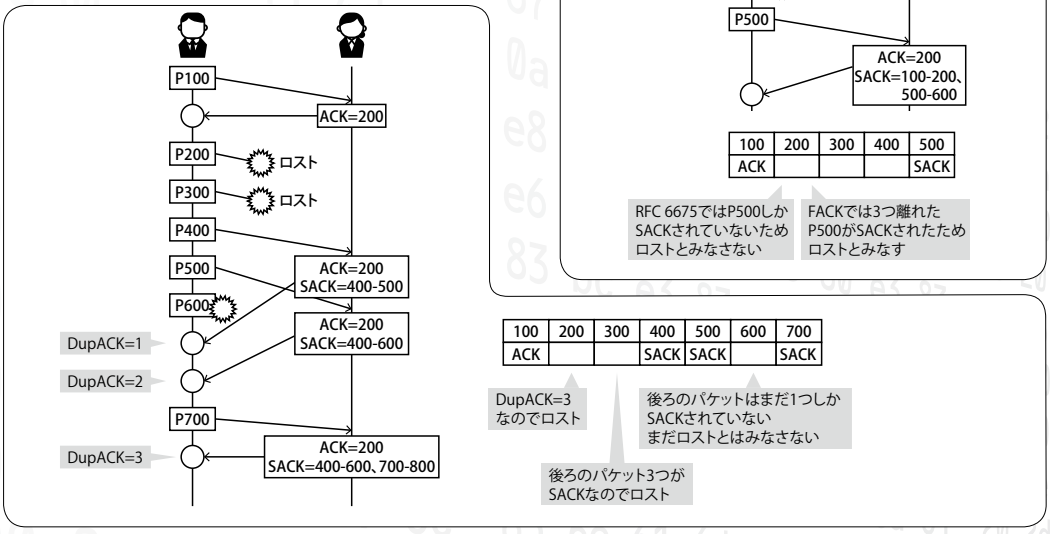


FACK：Forward Acknowledge

FACK（Forward Acknowledgement）は、RFCとしては規定されていない、SACKを使ったロス検知アルゴリズムです。Linuxにも実装されており、Linux 4.10まではデフォルトで使われるようになっていました。

FACKでは、まだACKされていない各パケットに対して、それ以後で、かつDupThresh以上離れたパケットがSACKされた場合にパケットロスとみなします。具体的に見てみましょう。**図4**のようにP100-500を送ったところ、P200-400が失われたとします。このときP500に対して、“ACK＝200、SACK＝100-200、500-600”が返ってきます。ここでP200のロスト判定を考えてみましょう。先ほどのRFC 6675のアルゴリズムであれば、まだP500しかSACKされていない（SACKed＝1＜DupThresh＝3）ので、P200はロストとみなされません。一方で、FACKであれば、P200と今回SACKされたP500とは3パケット分離れているのでロストされたとみなされます。このようにFACKは、RFC 6675よりもシンプルで、アグレッシブにロスト判定を行うアルゴリズムだと言うことができます。

▼図3　RFC 6675によるパケットロスの検知

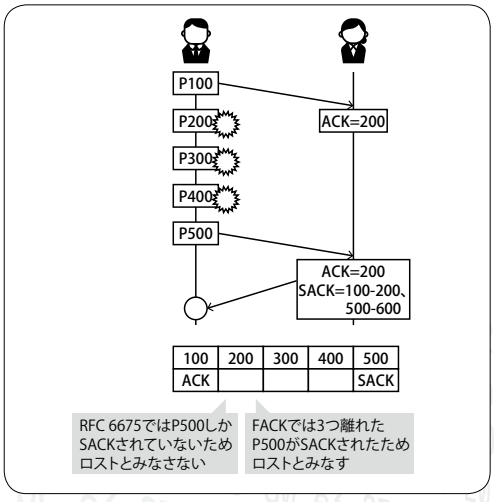


シンプルである反面、リオーダーに弱いという特性も持ちます。たとえば、先ほどと同じようにパケットがP100、P500と届きますが、ロストではなくリオーダーで、P200、P300、P400と届いたとしましょう。前述したように、P500によって“ACK＝200、SACK＝500-600”が返り、RFC 6675はP200をロストとみなさず、FACKはP200をロストとみなします。このパターンのパケットは従来のTCPのアルゴリズムでも再送なしに到着できるので、FACKはリオーダーには弱いアルゴリズムと言うことができます。しかしながら、リオーダーがない環境であれば、RFC 6675よりも早期にパケットロスを検出できるという、正確性とロスの早期発見とのトレードオフがあると言うことができます。

DupThreshの調整：RFC 5827とThin-Stream

ここまで、RFC 6675とFACKというSACKを使ってより賢くロスを検出する方法を紹介して

▼図4　FACKによるパケットロスの検知



きました。ここからはSACKに依存しない、特定の状況下でDupThreshを調整する改善方法を紹介します。

前述したようにDupThreshはデフォルトで3に設定されています。これにより、多少のリオーダーではパケットの再送が発生しないようになっています。一方で、ある種の状況下では、このDupThreshが「大き過ぎる」ことがあります。そこで一定条件を満たすことでDupThreshを小さくしたり、DupThreshを動的に調整したりする手法が提案されています。

そのような手法の1つとしてRFC 5827が規定するEarly Retransmitを見てみましょう(**図5**)。これは、送ることができるデータがもうなく、かつすでに送っていてACKされていないパケットが4つ未満である場合に、DupThreshを「すでに送っていてACKされていないパケット数−1」にするという方法です。

具体的に見てみましょう。送信側がP100からP400の4つのパケットだけを送り、P200が失われたとします。受信側はP300、P400にそれぞれ"ACK = 200"のDupACKを返しますが、これ以上パケットが来ないため最後の一押しとなるDupACKを返すことができません。通常のTCPでは、この後RTOが発生するのを待ってから、P200を再送することになります。

Early Retransmitが使える場合はどうなるでしょうか。まず、送信側はP300に対する"ACK = 200"のDupACKを見ます。このときは、まだP400という送信可能なデータがあるためEarly Retransmitは行われません。次にP400に対しても、"ACK = 200"のDupACKが返ってきます。ここでは、①これ以上送信できるデータはなく、かつ②送信してまだACKされていないパケット数がP200、300、400と3つで、4未満である、という2つの条件が成り立ちます。ここでEarly Retransmitが実行され、DupThresh=3−1=2と調整されます。すると、今のDupACKが2つ目なので再送の条件が満たされ、P200が再送されることになります。

RFC 5827をよりアグレッシブにしたThin-Streamという手法もあります。これはRFC 5827と同様に「送ったがまだACKされていないパケットが4つ未満」かつ、アプリケーション(またはシステム全体)のソケット設定によってその通信に"Thin-Stream"のフラグがついている場合に、1つのDupACKで再送を行うという方法です。

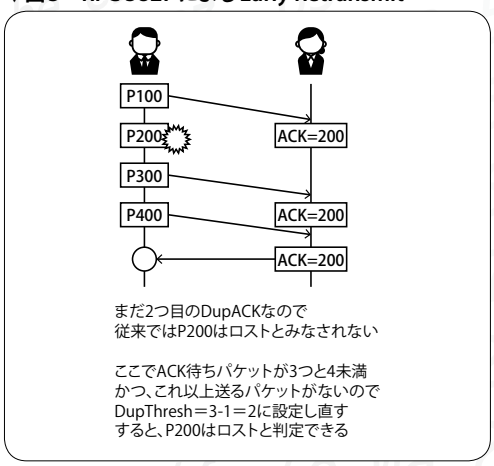
たとえば先ほど同様に、P100からP400が送られ、今度はP200とP300とが失われたとします。このあと、P400が到着し、"ACK = 200、SACK = 100-200"が返ってきます。ここで、まだACKされていないパケット数=3ですのでRFC 5827もThin-Streamも条件は満たしています。一方で、RFC 5827ではDupThreshが3−1=2に設定され、Thin-Streamでは1に設定されます。すなわち、RFC 5827では2つ以上のパケットが失われると再送が行われないのに対して、Thin-Streamではそのような場合でも再送が行われると言うことができます。もちろん、ここには、先ほど同様のトレードオフがあります。Thin-StreamはDupACKがあればすぐに再送を行うので、少しでもパケットの順序が変われればすぐに「無駄な」再送を行ってしまいます。



RFC 4653: DupThreshの動的調整

これまでのアルゴリズムはロス検出を早くす

▼図5　RFC 5827によるEarly Retransmit



ることに主眼を置き、DupThreshをひき下げるものでした。逆にリオーダーへの耐性に主眼を置いて、DupThreshをひき上げるアルゴリズムも提案されています。

RFC 4653は、TCPのNon-Congestion Events、すなわちリオーダーへの耐性を高めるアルゴリズムの提案です。この中では、DupThreshを(最小値3として)、送信したがまだACKされていないパケット数の半分までひき上げて、リオーダーへの耐性を高める方法が使われています。



RACKによるパケットロス検出

ここまで、オリジナルのNewRenoのロス検出、RFC 6675のSACKを使ったロス検出、FACK、RFC 5827のEarly Retransmit、Thin-Stream、RFC 4653のNCEと6つのロス検出の方法を紹介してきました。これらは、どれも早いロス検出とリオーダーへの耐性とのトレードオフがあり、両方を同時に満たすことができていません。

これらのアルゴリズムはどれもDupACKパケットの数によるロス検出を中心に据えていました。Linux 4.4で新しく導入されたRACK(Recent ACKowledgment)は、これらとは大きく異なりパケットのタイムスタンプによってロスを検出するという手段を中心に据えた新しいロス検出アルゴリズムです。このRACKにより、ここまで紹介したさまざまなアルゴリズムが置き換えられました。たとえば`tcp_time_to_recover()`という再送するかどうかを判定する関数では、以前は上記アルゴリズムに相当するものも含めて6つの判定条件を使っていたのですが、今では2つにまでその条件が減らされています。

また、このようにさまざまな手法が乱立している状況では、パラメータの調整が難しくなるという問題があります。すなわち、N個の機能があればおおよそ2^Nの組み合わせがあり、それらすべてを考慮してパラメータ・アルゴリズムを調整していくのは非常に厄介です。RACKが作られた動機の1つとして、これらさまざまな手法をRACKで置き換え管理・調整しやすくするというのもあります。

RACKでは基本的に、あるパケットPを送った時刻よりも十分あとに送ったパケットQがSACK(またはACK)されたときに、パケットPは失われたとみなします。この「十分あと」を判断する時間は`reo_wnd`(リオーダリングウィンドウ)という変数で管理されます。これはデフォルトでは1ミリ秒で開始し、パケットのリオーダーを検出した場合、`reo_wnd`を通信中での最小RTTの4分の1に設定します。すなわち、ここに設定された時間分だけ、パケットのリオーダーを許容することになります。

具体的にRACKの動きを見てみましょう。送信側はパケットPとQとを送り、Pはロストして、QへのACK(SACK)が返ってきたとします。ここで**図6左上**のようにPの送信時刻Tpが、Qの送信時刻Tqより`reo_wnd`時間、たとえば1ミリ秒以上前であれば、十分な時間が経ったのにパケットPが返ってきていない、ということでパケットPをロストしたとみなします。

では、ここで逆に**図6右上**のようにPとQとの送信時刻の差が`reo_wnd`より小さい場合はどうなるでしょうか。このときは、先ほどの条件が満たされていないので、まだリオーダーして到着する可能性を考えてロストしたとはみなしません。ここでACKの到着がもう少し後であれば、Pをロストしたとみなせることに注目してください。正確に言えば、あと`reo_wnd-(Tq-Tp)`時間だけ後にACKが届いていればPをロストとみなせたはずです。そこでRACKでは、この後者の場合にPをロストとみなせる時刻に動くタイマーを設定し、Pのロスト判定を行います。

従来のアルゴリズムでは、DupACKの数(またはSACKされたパケットの数)が中心となっていたため、新たなパケットが届かない限り、ロストの判定が変わることはありませんでした。一方で、RACKはパケット送信時点からの経過時間を中心としているので、一度でもACKを受

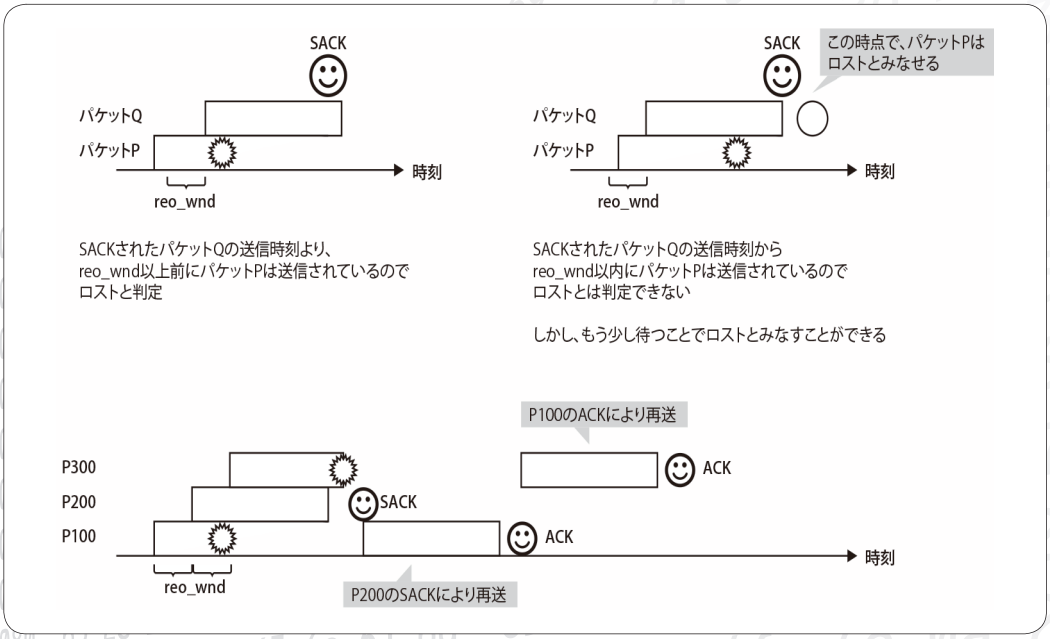
け取ればタイマーによってほかのパケットのロスを判定できるという強みがあります。ここで上記の2つのケースをまとめると、RACKでは、あるパケットが、最後にACKされたパケットのRTT + reo_wnd以上ACKされなければロストとみなす、と言うことができます。

また、RACKの特徴である時間差による判定は再送に対するACKへの挙動にも影響してきます。再送が失われるケースでのSACKベースのロス検出(RFC 6675)との違いを見てみましょう。たとえば、P100-300と3つのパケットをそれぞれ**reo_wnd**の時間以上あけて送り、P100、300の2つが失われたとします(**図6下**)。P200に対して、送信側へ"ACK = 100、SACK = 200-300"が返ってきます。P100は、P200よりも**reo_wnd**以上前に送られているので、P100は失われたと判定され再送されます。このとき、まだP300は失われたとは判定されません。このあと、P100の再送に対して、"ACK = 300"が返ってきます。このACKによって、P300も失われたと判定され、再送されることになります。一方でRFC 6675では、P200だけではSACKされたパケットがDupThresh分ではないのでまだ、P100は失われたとはみなされません。では、ここでThin-Streamが使われていたらどうでしょうか。たしかにP100については、P100以降でSACKされたパケットP200が存在するのでロストとみなされます。しかし、その一方でP300についてはシーケンス番号の順序でP300以降にACK/SACKされたパケットがないために、P300はP100の再送のACKがあっても失われたとはみなされないことになります。

最後にRACKのパフォーマンスを見ていきましょう。RACKのInternet-Draftによれば、RACKとTail Loss Probe(TLP)[注2]、RFC 3517(SACKベースのロス検出)を併用することでRFC 3517に対して25%、リカバリのレイテンシが改善していると言われています。さらに、RFC 3517を切ってもほとんどレイテンシが変わらない(0.02%改善)ことから、従来のロス検出であるRFC 3517を安全に置き換えられることがわかります。SD

注2) https://tools.ietf.org/html/draft-dukkipati-tcpm-tcp-loss-probe-01

▼図6　RACKの動き



Unixコマンドライン探検隊
Shellを知ればOSを理解できる

Author 中島 雅弘(なかじま まさひろ) ㈱アーヴァイン・システムズ

好奇心が湧くコマンドを探しましょう。Unixにはゲームやジョーク、いったい何の役に立つのだろうというコマンドも潜んでいます。そうしたコマンドを掘り出し、有用性を確認しながら、背景にある技術に迫ります。



第18回 ゲーム＆ジョーク特集(その3)



Unixの中のオアシス



Message Of The Day -- motd

Unix系システムには、ログイン時のメッセージを表示するしくみがあります。/etc/motdに管理者権限で、メッセージ(テキストファイル)を入れておけば、loginコマンド[注1]が成功して、シェルが起動される前にメッセージを表示します。日本語に対応した環境なら、日本語のメッセージも表示できます。

システム共通で表示されるmotdですが、「私には邪魔(じゃま)」という利用者はホームディレクトリに、.hushloginというファイルを作っておくと、メッセージを表示せずにログインできます。



banner -- print large BANNER on printer

bannerは看板を作るコマンドです。入力したASCII文字のテキストを大きな文字に変換して表示・印字します。bannerはとても古くからあるコマンドです。macOSのbannerコマンドは、最近ではほとんど目にしなくなった13インチのラインプリンタでの印字に合わせて、132文字幅がデフォルトです。幅といっても、90度回転した状態で1文字＝1ドットとして文字を作りますので、高さが132ドットです。

macOSのbanner(bashのバージョンは4.4.12(1)-releaseで確認。動作結果は、自身で確かめてみてください)

```
$ banner "Hello!"
```

Ubuntuなどの Debian系では、sudo apt install sysvbannerで導入できますが、このパッケージはmacOSのものとは動作が異なります。

Ubuntuのbanner

```
$ banner "Hello!"↵
#     #                                        ###
#     #  ######  #       #       ####    ###
#     #  #       #       #       #    #  ###
#######  #####   #       #       #    #   #
#     #  #       #       #       #    #
#     #  #       #       #       #    #   ###
#     #  ######  ######  ######   ####    ###
```

Linux版のbannerは、単語の最初の10文字を画面上で大きなアスキーアートとして表示するコマンドです。スペースで切って複数の単語を並べることができます。



fortune -- おみくじクッキー

合衆国の中華料理屋の会計時にたいてい出てくる、おみくじクッキー。これをUnix上に実現したのがfortuneです。実行すると、おもしろそうな言葉が表示されます。標準で用意されているメッセージは英語で、ちょっと難解な表現もあります。また、一部のメッセージには、人を不愉快にする内容を含むので、これらの出

注1) loginコマンドによらないログイン処理では表示されません。

力を抑止するオプションがあります。

インストールは、macOSではbrew install fortune、Debian系のUbuntuなどはsudo apt install fortune-modでできます。.bash_profileや.profileの最後の行にfortuneを加えておけば、ログインの都度「無用な」メッセージを見て楽しむことができます。;-)

STEP UP! .profile、.bash_profile、.bashrcなど

シェル[注2]の起動時に読まれるのがホームディレクトリにある、.profile、.bashrc[注3]などです。シェルの初期設定を定義できる便利なしくみです。システム共通の設定は、/etc/profileや/etc/bash.bashrc、利用者の個人設定は、~/.profileや~/.bashrcに定義しておきます。

実行順序は**図1**のとおりです。単純な端末と、リモート端末やウィンドウシステム上の端末での動作の違いにきめ細かく対応できるように、ログイン端末と非ログイン端末で実行されるファイルが異なります。さらに同じbashでもシステムによってファイル名なども異なることがあり、初心者にはとてもわかりにくい部分です。

使い分けの基準は、端末設定（stty、tsetなど）、環境変数、ログイン時に一度しか実行したくないコマンドは.profileに書きます。ほかのシェル変数やオプション（history、ignoreeofなど）、alias、関数などは.bashrcに書くのが原則とおぼえておきましょう[注4]。そして、シェルの起動時に無用な対話処理や負荷がかかる処理などは、これらに記述しないようにします。

さらにログアウト時には、~/.bash_logout、/etc/bash.bash_logout[注5]の順で実行されます。history情報などのセーブは、ここで処理します。

fortuneのデータファイルの作り方

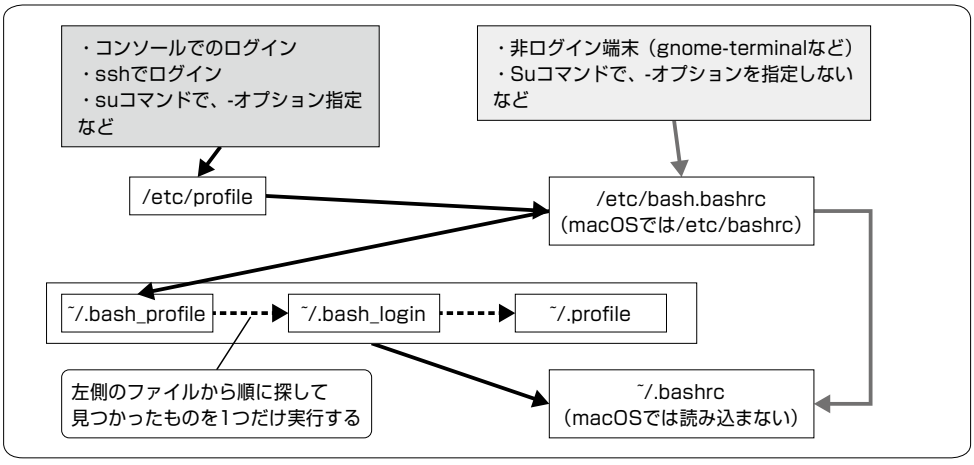
fortuneは、メッセージの「ネタ」ファイルを指定できるので、自分のベストプラクティス集などを登録しておくと実用性がアップします。著者のオフィスでは、古今東西の名言・迷言、社内ルール、DevOpsにおけるベストプラクティスなどを用意し、社内用Chat Botがfortuneを実行し、適当に説教を唱えてくれています。:-)

ファイルの場所は、Debian系（Ubuntu）は/usr/share/games/fortunes、macOS（brew環境）は/usr/local/Cellar/fortune/9708/share/games/fortunesなどです（正確な場所は、man fortuneで確認しましょう）。

独自のメッセージの作り方は、上記ディレクトリに、次のようなテキストファイルを作ります。

注2）ここではおもに、Ubuntu 16.04のbashにフォーカスします。
注3）rcは、Run Commandの意味です。
注4）ここに記した原則は、csh以来のやり方です。
注5）Ubuntu 16.04の英語のmanpageには、bash.bash.logoutと記述があるが間違いのようです。

▼図1　スクリプトファイルが読まれる順序
※Ubuntu 16.04のbashでの例（スクリプトファイル中で、次のファイルを読み込んでいる）



fortuneデータファイル「joe」
```
明日のためのその1: うつべし
%
明日のためのその2: うつべし、うつべし
%
明日のためのその3: うつべし、うつべし、ウツベシ!
%
```

strfileコマンドを使って、ランダムアクセスできるファイルを作ります。strfileコマンドは、上記のデータファイルのように、セクションを「%」記号1つで区切って解釈します。

```
$ strfile joe joe.dat
```



cowsay

メッセージをアスキーアートキャラに話させる**cowsay**は、プロビジョニングツールとして人気の高いAnsibleのPlaybook実行時にも見ることができるのでご存じの方もいるかもしれません。なんとなくおもしろいけど、役に立つのでしょうか。;-)

macOSならbrew install cowsay、Debian系であれば、sudo apt install cowsayで導入できます[注6]。牛以外にもいろいろなキャラクタがあり、-lオプションでキャラクタのリストが得られます。キャラクタは、-f <キャラクタ名>と指定します。

STEP UP! fortuneのお言葉をcowsayでいろいろなキャラクタにしゃべらせる技
```
$ fortune | cowsay -f $(cowsay -l | awk
'NR>1{cs = cs " " $0} END{n=split(cs, c);
srand(); print c[int(rand()*n+1)]}')
 ______________________________________
< Stay away from flying saucers today. >
 --------------------------------------
        \    ,-^-.
         \   !oYo!
          \ /./=\.\______
               ##        )\/\
                ||-----w||
                ||      ||

               Cowth Vader
```

Ubuntu 16.04では、cowthinkがcowsayへのシンボリックリンクとして作られます。cowthinkは、「考えている」吹き出しになります。



say、espeak -- コンピュータに発声させる

macOS限定のコマンド**say**は、テキストを音読させるコマンドです。引数で与えた文字列を読み上げます。コマンドを実行するPCのオーディオ出力をオンにして、コマンドを実行してみましょう。

```
$ say "おはよう"
```

sayは、日本語だけでなく、ほかの言語をしゃべる者も登録されています。どんな声があるのか見てみましょう(**図2**)。

en_のロケールが英語でしゃべる人です。US(米国)、GB(英国)、AU(オーストラリア)、IN(インド)いろいろありますね。sayは、-vオプションで読み手を指定、-f <ファイル名>で読み上げるファイルを指定、読み上げのファイルも文字列も指定しなければ標準入力からのテキストを読み上げます。

連載第17回(本誌2017年9月号)のテキスト処理で取り上げた題材ハムレットを使って、どの音声がハムレット役に適しているのか、オーディションを開催しましょう(hmlt.txtがハムレットの英文テキストファイル名)。

ハムレット役のオーディション
```
$ for actor in $(say -v ? | egrep en_ | awk
'{print $1}') ; do sleep 2; echo ${actor};
say -v ${actor} ${actor}; sleep 1; (egrep -C
4 -i "not to be" hmlt.txt | say -v ${actor})
; done
```

少々長いワンライナーですが、探検隊の読者なら、順番に読んでいけば理解できるはずです。音声の一覧から、英語をしゃべる音声を抽出し、それぞれの音声でハムレットの「成すべきか、成さざるべきか……」の部分を読み上げさせています。

「日本語も試したい」という方は、中原中也の

注6) cowsayの日本語対応：Ubuntuの場合、カラム指定の「-W」は文字数でワードラップするので、画面上の半角文字のカラム数ではありませんが、日本語文字もきちんと表示できます。macOSのものは、日本語の表示はできますが文字数やワードラップの対応がないようです。

▼図2　sayでしゃべる声の一覧

```
$ say -v ?
Alex                en_US    # Most people recognize me by my voice.
Alice               it_IT    # Salve, mi chiamo Alice e sono una voce italiana.
Alva                sv_SE    # Hej, jag heter Alva. Jag ar en svensk rost.
Amelie              fr_CA    # Bonjour, je m'appelle Amelie. Je suis une voix canadienne.
Anna                de_DE    # Hallo, ich heise Anna und ich bin eine deutsche Stimme.
 ...略...
```

▼図3　wコマンドの出力例

```
 11:10:04 up 5 days, 15:58,  4 users,  load average: 0.71, 0.59, 0.56
USER  TTY      FROM             LOGIN@   IDLE    JCPU    PCPU WHAT
masa  tty7     :0               木19     5days   10:35m  0.52s /sbin/upstart --user
masa  pts/5    119.106.24.104   火16     17:47m  0.16s   0.16s -bash
masa  pts/21   192.168.2.145    火19     1.00s   0.17s   0.02s w
bot   pts/23   192.168.2.145    11:10    3.00s   0.13s   0.13s -bash
```

詩“汚れつちまつた悲しみに……”“サーカス”などをOtoyaとKyokoに読ませてみましょう……。

Linuxにも**espeak**というコマンドがあります[注7]。macOSに比べると、役者がちょっとぎこちないですね。:-) 今のところespeakに、日本語をしゃべる役者はいないようです。

ハムレット役のオーディションLinux版

```
$ for actor in $(espeak --voices | awk '$2 ~ /^en/{print $4}'); do sleep 2; echo ${actor}; espeak -v ${actor} ${actor}; sleep 1; (egrep -C 4 -i "not to be" hmlt.txt | espeak -v ${actor}) ; done
```



誰が何をしている?……話しかけてみるか

Unixは、マルチユーザ・マルチタスクのOSです。自分が作業しているホストに、ほかに誰かログインしていないか、何をしているのか気になりませんか。また、ログインしている利用者にメール以外の方法で、メッセージを送ってみませんか。



w -- show Who is logged on and what they are doing -- 誰がログインして、何をしているか

wコマンドは、誰がログインしているか確認できるコマンドです。どのttyに接続して、いつからログインしているか、今何を実行しているかもわかります(**図3**)。



who -- show WHO is logged on -- 誰がログインしているか

wと類似のコマンドに**who**があります(**図4**)。



last -- show a listing of LAST logged in users

w、whoは、現在ログインしている利用者が対象でした。ログアウトしてしまっている利用者のログイン記録を確認するには、**last**コマンドを使います(**図5**)。



lastcomm -- show LAST COMMands executed in reverse order

lastcommは、最後に実行したコマンドの履歴を表示します(**図6**)。acctというログを参照



▼図4　whoコマンドの出力例

```
masa tty7     2017-06-08 19:12 (:0)
masa pts/5    2017-06-13 16:55 (119.106.24.104)
masa pts/21   2017-06-13 19:51 (192.168.2.145)
bot  pts/23   2017-06-14 11:10 (192.168.2.145)
```

注7) Debian系Ubuntuなどは「$ sudo apt install espeak」で導入できるはずです。

しますので、導入されていないシステムでは、apt install acctなどで導入してください。macOSは、sudo touch /var/account/acct; sudo accton /var/account/acctでアカウントログの記録を開始すれば、lastcommが使えるようになります。LinuxでもmacOSでも共通のacctonコマンドは、アカウント記録をオン／オフできるコマンドです。

w、who、last、lastcommは、いずれもシステム内での利用者の行動を追跡することができます。システム管理者側や一般の利用者にとって、システムに不審な痕跡が残っていないかなどを確認できる有益なツールです。一方、悪意のある利用者にも、格好の情報収集ツールです。悪事の行動記録を残したくない利用者は、ログを確認できないように改ざんします。私たちは、不用意にアクセス権限を提供しないようにすることや、システム上の不自然な点への注意を払っておく必要があります。



wall -- Write a message to ALL users, talk -- TALK to another user, write -- send a message to another user（WRITE to another user）

ログインしているすべての人にメッセージを送るのがwallコマンドです。リモートシステムには表示されませんが、システム管理者がメンテナンスの情報を伝えるときなどにとても重宝します。ジョークコマンドというより、実用性のあるコマンドですが、使い方によっては人に迷惑をかけることもできます。:-)

mesgコマンドの設定が、「y」のユーザ全員に伝えます。mesgは、ほかのユーザからのメッセージを受け付けるかどうかを設定します。自分自身が複数の端末を開いている場合は、mesgの設定にかかわらずメッセージが表示されます。また、スーパーユーザ(root)は、mesgの設定にかかわらず、メッセージを送れます。また、wallコマンドはシステムによって、メッセージストリングを引数に取れるものと、ファイル名のみしか受け取らないものがあります。

メッセージを送る
```
$ echo 'Hey, wake up!' | wall
```

mesgの確認と各端末での表示
```
$ mesg↵
y

$
Broadcast message from bot@my-ubuntu
(pts/22) (Tue Jun  6 18:27:49 2017):

Hey, wake up!
```

wallと同様talk[注8]、writeコマンドは、相手

注8）接続するには対象のマシンでtalkdを起動しておく必要があります。

▼図5　lastコマンドの出力例
```
$ last -4↵
bot      pts/23       192.168.2.145    Wed Jun 14 11:10 - 11:11  (00:01)
masa     pts/23       192.168.2.145    Tue Jun 13 20:07 - 20:08  (00:00)
masa     pts/21       192.168.2.145    Tue Jun 13 19:51   still logged in
masa     pts/21       192.168.2.145    Tue Jun 13 19:14 - 19:15  (00:00)

wtmp begins Thu Jun  1 12:46:17 2017
```

▼図6　lastcommの出力例
```
$ lastcomm | head -5↵
command-not-fou        masahiro pts/21     0.26 secs Wed Jun 14 11:20
ls                     masahiro pts/21     0.00 secs Wed Jun 14 11:20
sshd             S     root     __         0.00 secs Wed Jun 14 11:20
lastcomm               masahiro pts/21     0.00 secs Wed Jun 14 11:20
sshd             S     root     __         0.03 secs Wed Jun 14 11:20
```

もしくはttyを指定してメッセージを送信します。talkは対話チャット型、writeはワンタイムのメッセージです。これらも、mesgの状態を見て、送れる、送れないが判断されます。

STEP UP! gnome-terminalなどでmesgの設定にかかわらずwall、talk、writeを受信できないことについて

mesgの設定にかかわらず、Ubuntu 16.04のgnome-terminalなど一部の端末は、wall、talk、writeを受け付けません(14.04では表示できます)。少し、踏み込んで調べてみましょう。

Ubuntu 16.04のgnome-terminal以外では、多くのLinuxのGUI標準の Ctrl + Alt + F1 ～ F6 による物理tty[注9]は出る、sshでログインしているのも出る、xtermは出る、といった具合です。出る端末と出ない端末で、パーミッションとmesgの状態を確認してみましょう。

GUI(gnome-terminal)で開いている端末

```
$ tty
/dev/pts/22
$ ls -l /dev/pts/22
crw--w---- 1 masa tty 136, 22  6月  6
18:12 /dev/pts/22
$ msg
is y
```

sshでログインしている端末

```
$ tty
/dev/pts/23
$ ls -l /dev/pts/23
crw--w---- 1 masa tty 136, 23  6月  6
18:15 /dev/pts/23
is y
```

どちらも、mesgは「y」で、ttyグループに書き込み可能になっています。不思議ですね。

Unix系のシステムには、utmpとwtmpというログイン状態を記録しているファイルがあります[注10]。端末が、utmpに書き込みをしていないとwで確認しても、その端末はリストアップされません。

また、wallやwriteは、utmp情報を参照して、そこに記録がないものはメッセージを送信する対象から外します。ということで、gnome-terminalなどのようにメッセージを送れないのです[注11]。

utmpに書き込むか、書き込まないかはプログラムによります。たとえば、ウィンドウマネージャ、ttyを持たないバッチコマンド、サービスデーモンなどをすべて記録するのは、ログイン記録というutmp、wtmpの目的から外れています。Ubuntu 16.04のgnome-terminalは、開発者の考えで[注12]utmpへの書き込みをしていません。ということで、gnome-terminalに対しては、wallなどでの書き込みができないのです。

今回の技術が活躍するところ

コンピュータにテキストや音声で利用者に情報を伝える方法を確認しました。さまざまなデバイスが共通のI/Oインターフェースで扱えるところはUnixの大きな特徴です。これまでに学んだテキスト処理やシェルの技術を使うことで、複数のコマンドを効果的に連結できました。

また、近隣の利用者間での単純なメッセージ伝達のしくみや利用者の状態を見るコマンドでは、Unixがマルチユーザ OSであることを再確認し、一般利用者のアクセス権限と管理者権限の扱い、システム記録(ログ)の知識がセキュリティ視点において重要であることを体験しました。

普段使わないような、コマンドやその周辺のしくみにも、重要な情報がたくさん埋まっています。

次回について

次回は、ssh(その3)を予定しています。SD

注9) 本誌2017年5月号 第13回参照。

注10) Linux系のシステムでは、だいたい/var/run/utmp、/var/run/wtmpにあります。macOSでは、utmp、wtmp、lastlogは/var/run/utmpxに統合されています。

注11) writeもwallも、本体はC言語で350～450行程度の短いプログラムです。興味があれば、ソースを読んでみると良い勉強になると思います。

注12) ディスプレイマネージャ配下で動作している端末で、これをログインシェルと考えるのか、非ログインシェルと考えるのかは悩ましいところです。

今回の確認コマンド

【manで調べるもの
(括弧内はセクション番号)】

motd(5), login(1), banner(6), fortune(6), bash(1), cowsay(6), awk(1), w(1), who(1), last(1), lastcomm(1), accton(8), acct(5), wall(1), write(1), talk(1), talkd(8), utmp(5)

[Linuxのみ]banner(1), espeak(1)

[macOSのみ]say(1), utmpx(5)

NO.72

Monthly News from jus
Japan UNIX Society

小林 峻　KOBAYASHI Takashi

地域を超え、ますます激しさが増すシェル芸勉強会

今回は、7月に行ったシェル芸勉強会の報告をお送りします。

シェル勉強会／シェルワンライナー勉強会

■jus、USP友の会共催
シェル／シェルワンライナー勉強会
【講師】鳥海 秀一（USP友の会）、
上田 隆一（USP友の会）
【日時】2017年7月1日（土）10:00～18:00
【会場】さくらインターネット㈱
西新宿セミナールーム

「jus・USP友の会共催 第29回激しいシェル芸勉強会」が、7月1日に西新宿のさくらインターネットで開催されました。東京の本会場のほか、大阪、福岡には有志によるサテライト会場もあり、東京は33人の参加がありました。今回も恒例となった、午前の部（初心者向け講座）、午後の部（問題を解く本編）、懇親会（LT大会）の3部構成で、朝から晩まで黒い画面に向かってシェルと戯(たわむ)れました。

■午前の部

午前の部はスクール形式の勉強会で、講師はUSP友の会の鳥海さんです。前回（4月）の勉強会で「Perlの正規表現」が便利という話題が出たので、次回の勉強会の午前の部のお題として提案したところ、言い出しっぺが勉強して講師をやることになったそうです。しかし、Perlについて触れずにいきなり正規表現というわけにもいかないので、今回は「Perlワンライナー入門」講座として、正規表現は次回に持ち越しになりました。

さて、今回の内容ですが、ワンライナーでPerlを使う際に便利な特殊変数や豊富な起動オプションを中心に解説がありました。Perl独特の特殊変数や省略など多彩なイディオムがたくさんあり、普段はAWKでなんとかしてしまうシェル芸人も、Perlの多様性に圧倒されたと思います。「Perlは読むもんじゃない」とよく言われるようですが、打ち捨てることの多いシェル芸と相性が良いのかもしれません。

ワンライナーを覚えるコツは「何度も使うこと」だそうで、ハンズオンを交えながら行われ、会場は黙々とワンライナーを打ち込んだり、試して結果に驚いたりする姿が多くありました。講師は自称Perl歴0年とのことでしたが、たいへんわかりやすくマニアックな講座でしたので、次回も楽しみです。

■午後の部

午後の部は5～6人のグループに分かれ、初心者



図1　シェル芸bot

からベテランまで仲良く問題に取り組みます。問題の出題と解説はUSP友の会の上田さんです。今回の出題は、「データのJOIN」「パズル」「巨大文字」の3テーマ全8問でした。午後は各会場やリモート参加のみなさんが解答や解説を、「#シェル芸」を付けてTwitterへ投稿します。とくに今回はタイムライン上のシェル芸を実行してくれるシェル芸bot（@minyoruminyon、**図1**）の登場により、いつも以上に盛り上がりました。

「データのJOIN」問題では、学生の出席状況（架空）などを例に、データの結合や集計をする問題が出題されました。大学教員の上田さんが、普段やっている実用的な問題が元ネタです。このような問題では、AWKやjoin(1)が使われることが多いのですが、会場からはGNU datamash(1)を使った解答なども飛び出し、さまざまな学びがありました。

「パズル」問題では、文字を転がす問題などが出題されました。「文字を転がす」の意味がわかりにくいですが、三角形に並んだ文字列を図形のように転がす問題です（**図2**）。このテーマでは行列を転置するBSD rs(1)が大活躍しました。また、GNU sort(1)の-gオプション（一般的な数値としてソート）など細かいオプションの違いを知ることもできました。

最後は、「巨大文字」問題です。文字列を巨大化したアスキーアートをコンソール上に描画するtoilet(1)というコマンドがあります（**図3**）。そのtoilet(1)で書かれた巨大文字列を題材に、HTMLの文字実体参照から変換したり、ぎゅっと縮めたりと多彩な問題が出題されました。

次のファイルの中身について、

```
$ cat triangle
      1
     3 9
    7 a 6
   8 4 2 5
```

次のように右に転がしてください。
できる人は awk を使わないでやってみましょう。

```
     8
    4 7
   2 a 3
  5 6 9 1
```

図2 「文字を転がす」問題
（出典：URL https://blog.ueda.tech/?p=9870）

■懇親会

勉強会終了後は、さくらインターネットの会場をそのまま借りて、ビールを片手に簡単な自己紹介とLT大会が行われました。今回はLTの中継だけでなく、東京会場には関西からの参加者が、大阪会場には関東からの参加者がおり、いつも以上に地域の一体感がある懇親会になったと思います。とくに東京会場では容赦なく環境を破壊して研究する大阪の熱い雰囲気を感じられました。

ほかにも、フラクタル図形を描画するクワインシェル芸やJavaScriptシェル芸、秘密分散ストレージに関する学会発表ダイジェストなどさまざまな発表があり、いつも以上に熱い議論が交わされました。シェル芸勉強会のLT大会はテーマの幅も広く、軽いノリでとても濃い発表が多いのでとても楽しいです。

■まとめ

シェル芸勉強会は、シェルを便利に使いこなすというのが勉強会の趣旨の1つですが、シェルに限らず何か新しい学びや気づきがあれば良いものです。今回も多種多様な人たちが集まり、非常に楽しく学びある勉強会になりました。勉強会の様子が気になる人は、Twitterのまとめや参加者のブログ[注1]なども併せてご覧ください。SD

```
$ toilet SD

  mmmm  mmmm
 #"   " #   "m
 "#mmm  #    #
     "# #    #
 "mmm#" #mmm"
```

図3 toiletコマンドの実行例

注1） URL https://blog.ueda.tech/?p=9922

あなたのスキルは社会に役立つ

2011年3月11日の東日本大震災発生の直後にHack For Japanは発足しました。今後発生しうる災害に対して過去の経験を活かすためにも、エンジニアがつながり続けるためのコミュニティとして継続しています。防災や減災、被災地の活性化や人材育成など、「エンジニアができる社会貢献」をテーマにした記事をお届けします。

第70回
盛り上がる技術系同人活動に乗っかれ! 「Code for 同人誌」活動報告

●Hack For Japanスタッフ　小泉 勝志郎（こいずみ かつしろう）　Twitter @koi_zoom1

はじめに

同人誌。みなさんこの言葉からどういう連想をするでしょうか？　オタク、マンガ、BL、18禁……。そういう想像をする方も多いでしょう。しかし、いま熱くなってきているそれ以外のジャンルがあります。それが技術系同人誌です。

今年のDevelopers Summit 2017では技術系同人誌即売会コーナー「DevBooks」が設置されていますし、技術系同人誌即売会「技術書典」では多くの入場待ち行列ができるほどの盛況でした。プログラミング言語関係の同人誌も多いですがそれだけではありません。自作ハードウェアやその本を出している方もいれば、大昔のハードウェアを徹底的にハックする本を出す方も。

この技術系同人誌のビッグウェーブに乗るしかないでしょ！ということで、同人誌を作成してサークルとして参加してきました。2017年4月9日の「技術書典2」、そして同人誌関係では世界最大のイベントである2017年8月11日の「コミックマーケット」。今回の記事ではその模様をお知らせします。

Code for 同人誌

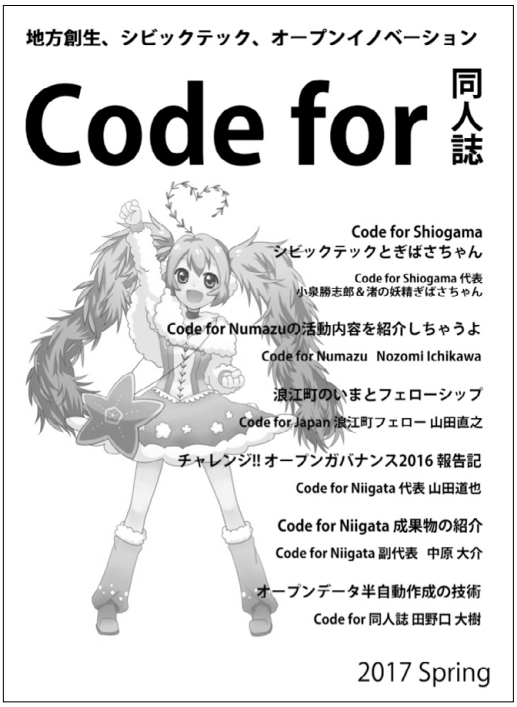
Code for 同人誌。筆者が参加したサークル名、そして本の名前でもあります（**写真1**）。

内容は「全国各地のCode for ○○」の活動成果をまとめたものです。Code for ○○は全国各地にあり、「地域課題をITで解決する」ことを目指しています。Code for Japanのブリゲイドという呼ばれ方もされますが、各団体上下なく独立した組織です。そのため活動内容も多岐にわたり、それぞれが全然違う活動をしています。

表紙のキャラクタはこの連載でも何度か出てきているCode for Shiogamaで作ったキャラクタ「渚の妖精ぎばさちゃん」！

このCode for 同人誌のVol.1を技術書典2で、Vol.2をコミックマーケットに出しました。販売形態は電

▼写真1　Code for 同人誌



子書籍。どちらもネットから1部500円で購入可能です[注1]。以降でその内容について説明していきます。

Vol.1

Vol.1は初めての作成ということもあり、手探り感が強いですが、その分バリエーションに富んだ内容となっています。塩竈、沼津、浪江町、新潟、そして同人誌オリジナルという布陣。イベントで一緒になるタイミングで執筆を依頼し、完成イメージをなかなか持てない中でのスタートでしたが、読み応えある内容に仕上がりました！

●シビックテックとぎばさちゃん

巻頭記事の執筆は宮城県塩竈市で活動するCode for Shiogamaの筆者です。内容は表紙にもなっているぎばさちゃんの誕生までのストーリー、そして人気が出る萌えキャラの作り方の説明となっています。

ここのポイントは「人気が出る萌えキャラの要素」というほかでは読めないノウハウが入っていること。過去にデブサミのオープンジャムで「萌えキャラ作成ワークショップ」を行った経験も反映した、萌えキャラ作成＆運営の虎の巻です。

ほかにも「渚の妖精ぎばさちゃん」誕生までのストーリーでは、東日本大震災後の塩竈の話、そして離島である浦戸諸島でのハッカソン「島ソン」の中からどうやってぎばさちゃんが生まれたのかを書いています。ITからスタートし、ITの枠を超えて活躍するぎばさちゃんのストーリーです。

●Code for Numazuの活動内容を紹介しちゃうよ

Vol.1の中で唯一のマンガです！　静岡県沼津市のCode for Numazuは非常に精力的に活動していて、その活動内容は幅広いものです（**写真2**）。今回はその幅広い活動の中でも「ラブライブ！サンシャイン!!」から派生した活動を紹介してくださいました。

「ラブライブ！サンシャイン!!」の舞台は沼津市内浦。まさしくCode for Numazuの活動場所。聖地巡礼に来たアニメファンも多く来ることが考えられるので、地元民の理解を深めるために「使っている人のことを考えた聖地マップ」を作った体験談！

●浪江町のいまとフェローシップ

東日本大震災での福島第一原発の事故で全町域が避難対象となった福島県浪江町。Code for Japanにはフェローシップという、メンバーを自治体の職員として派遣するしくみがあり、浪江町にもフェローシップを派遣しています。2017年6月までフェローシップとして浪江町に勤務した山田直行さんからの活動報告です。

浪江町は全町域が避難対象となったため、震災以降町役場も浪江町から二本松市内に移って仮役場が設置されていました。ちなみに2017年3月31日には、浪江町の町の東側20％ほどが避難指示を解除され、町役場も浪江町内にほとんどの機能が移されています。

町民が避難でバラバラになった自治体の中でのタブレットを用いたきずな再生・強化事業の話や、エンジニアとして被災自治体に勤めることなど非常に貴重な記録です。

▼写真2　Code for Numazuの活動報告マンガ



注1　https://code-for-doujin.booth.pm

●チャレンジ!!オープンガバナンス2016報告記

Code for Niigataからは2本の記事が。「チャレンジ!!オープンガバナンス2016」は各種公開データを用いた地域課題解決を促す目的のコンテストで、Code for Niigataは新潟大学大串ゼミとの混成チームでの出場体験記です。なんと連携体制賞を受賞しています。

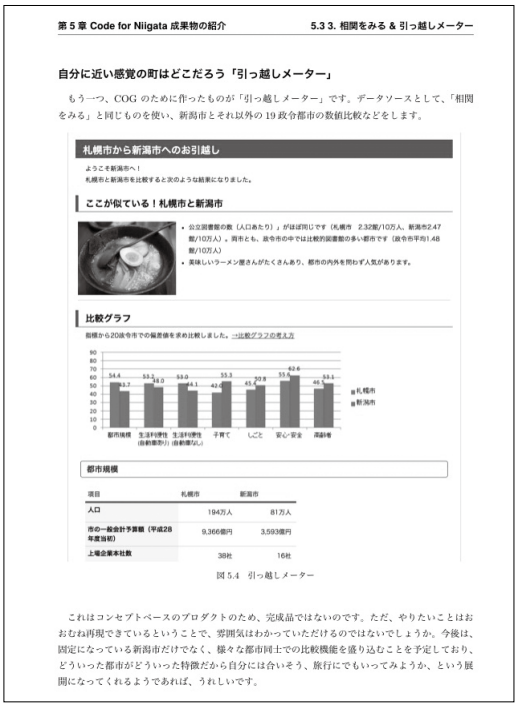
●Code for Niigata成果物の紹介

Code for Niigataが活動の中で作った成果物が紹介されています。次の4本です。

- バスの時刻表検索サービス（公開停止中）
- pics：画像に位置情報やコメントを付けて地図上に表示するシステム
- 相関を見る
- 引越しメーター

「相関を見る」と「引越しメーター」（**写真3**）は「チャレンジ!!オープンガバナンス2016」で受賞につながった成果物とのことです！

▼**写真3　引越しメーターの記事**



●オープンデータ半自動作成の技術

技術系同人誌らしいソースコードも出てくる記事です。Facebookのオープンデータ界隈で有名（？）なハッシュタグ「#オープンデータソフトクリーム」。このデータを自動生成しようというものです。

Vol.2

Vol.2では、Code for Japan代表の関治之さんから熱い序文ももらえました。

●ぎばさちゃん「ふくいち」へ福島第一原発見学記

先日縁あって福島第一原発を見学する機会を得ました。バスの中からの見学ですが、実際に発電所構内に入り原子炉建屋の近くへも行っています。そこでわかったのは確実に作業は進んでいるし、状況もよくなっているということ。そして、自分自身東日本大震災被災者でありながら、原発関係の知識があまり更新されてなかったことです。

そこで、ぎばさちゃんにマンガで語ってもらうという手法を取りました（**写真4**）。マンガの冒頭に、わざわざ東北物産展の場で「東北の海のものなんか売ってるのか？　放射能はどうなんだ？」と聞いてくる人がいますが、これは実際にそういうことがあったのです。ぎばさちゃんが主人公ですが内容は実話の組み合わせなのです。

●Code for NumazuとWikipediaTown

Code for Numazuで行っているWikipediaTownのレポートです。WikipediaTownは地域の情報をWikipediaに入力していくものです。沼津市に多くある古墳の情報を入力するために行ったことを、ノウハウとともに書かれています。余談ですが、マスコットキャラの古墳ちゃんたちがかわいいです！

●Code for Niigata～継続的な活動のために～

Code for Niigataからは継続的な活動のために取り組んでいることが紹介されました。「何のために活動をするのか？」「みんなで共有の『みんな』とは？」など代表の山田道也さんがCode for Niigata

▼写真4　福島第一原発見学記



▼写真5　オープンデータビジュアライズの記事

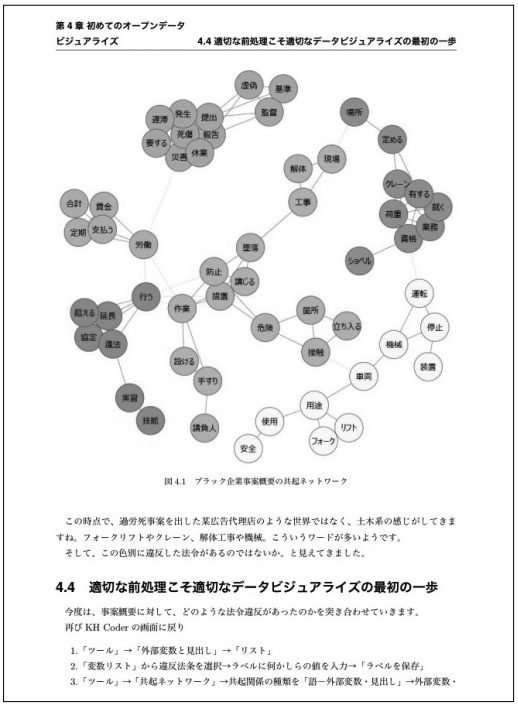


図 4.1　ブラック企業事案概要の共起ネットワーク

この時点で、過労死事案を出した某広告代理店のような世界ではなく、土木系の感じがしてきますね。フォークリフトやクレーン、解体工事や機械。こういうワードが多いようです。
そして、この色別に違反した法令があるのではないか。と見えてきました。

4.4　適切な前処理こそ適切なデータビジュアライズの最初の一歩

今度は、事案概要に対して、どのような法令違反があったのかを突き合わせていきます。
再び KH Coder の画面に戻り

1.「ツール」→「外部変数と見出し」→「リスト」
2.「変数リスト」から違反法条を選択→ラベルに何かしらの値を入力→「ラベルを保存」
3.「ツール」→「共起ネットワーク」→共起関係の種類を「語ー外部変数・見出し」→外部変数・

にかける熱い想いが伝わってくる内容です。

●初めてのオープンデータビジュアライズ

厚生労働省が出しているブラック企業リストを使った「ブラック・ブラック企業[注2]」。このサイトのリポジトリのデータを分析してビジュアライズしようというものです（**写真5**）。

そして即売会！

コミケのときはなんと実体化ぎばさちゃんに来てもらいました！（**写真6**）かわいいですね！

ただ、正直なところコミケのときより技術書典2のときのほうが販売部数は多かったです。コミケの混雑具合は想像を絶しますし、初めて同人誌を出すときは技術書オンリーイベントが良いかもしれません。余談ですが、コミケのときはあの叶姉妹のブースも付近にありました。遠目に見ただけですが、おきれいでしたよ！

▼写真6　実体化ぎばさちゃん



最後に

```
while (Japan.recovering)
    we.hack();
}
```

今後も技術系同人誌は盛り上がっていくと思います。良い意味で「自分が楽しいからその技術をやる！」。実際その熱を感じました。

Code for 同人誌では今後も地域の課題にITで取り組む姿を中心に楽しく取り上げていきます。みなさんオンラインでも購入できますのでよろしくお願いします。SD

注2　http://structure-and-representation.com/works/blackCorporate/

温故知新
IT むかしばなし

第70回

仮想86モード
～x86は、どのように16bit互換性を維持してきたのか

速水 祐（はやみ ゆう） http://zob.club/ twitter @yyhayami

はじめに

Intel社のCoreシリーズは現在、64bitモードの利用が広がりつつあります。x64で動作するOS上での、さまざまな工夫により、今までの32bit(IA-32)のアプリケーションの安定した動作が実現されています。

1980年代の後半に32bitのx86 CPUであるi80386が登場して、16bitのアプリケーションを32bitのモードから実行するためのi80386の機能が、同じように注目されていました。今回は、その互換機能である仮想86モードのお話をしましょう。

i8086からi80286、そしてi80386

1981年にIBM-PCにi8088、1982年にPC-9801にi8086が搭載され、MS-DOSが標準OSとなると数多くのx86のリアルモード(16bitモード：最大アドレス1MB)で動作する16bitアプリケーションが生まれました。

1984年にIBM-PC/AT、1986年にPC-9801VXがi80286を搭載して登場すると、i80286のリアルモード下で高速に16bitアプリケーションが動作するようになりました。i80286は、24bitのアドレスバスにより16MBのメモリアクセスができるとともに新たにプロテクトモードを備え、高機能なメモリ保護や特権リング[注1]によるタスクの特権レベルの制御などを行うことができたのですが、その高機能性を活かすOSがなかなか現れませんでした。やっと登場したIBMとMicrosoftが共同で開発したOS/2は、プロテクトモードの動作を基本としていたのですが、プロテクトモードで動作するアプリケーションはほとんどありません。今までの16bitアプリケーションを動かすためには、プロテクトモードで動作するOSは一端眠らせ、無秩序なリアルモードにしてから動かすしか方法がありませんでした。

さらにi80286は、保護機能が効かなくなるリアルモードへの切り替えは、リセット動作によって初期状態にする方法しかなく、切り替えに大きな時間を要することになったのです。となるとOS/2上での16bit MS-DOSアプリケーションの動作は実用的ではなくなり、結局、i80286搭載マシンは、高速なリアルモードで動作する高速な16bit MS-DOSマシンとしてしか生きる道はありませんでした。そこに登場するのが32bit x86であるi80386です。1986年にコンパックがIBMに先駆けてPC/AT互換機にi80386を搭載し、1988年に登場するPC-9801 RAにi80386が載ります。i80386は、i80286のプロテクトモードを32bitに拡張して4GBものアドレス空間にアクセスが可能になり、内部的なレジスタ関連も32bit化された本格的な32bit CPUでした。1988年当時もMS-DOSが主流でしたので、それをどのように動作させるかが問題となります。その解決策が仮想86モードだったのです。

注1）階層的な特権管理。

仮想86モードとは

仮想86モードは、プロテクトモードの1つのタスクとして動作し、**図1左**のように複数の仮想86タスクを構築できました。また、プロテクトモードのタスクとして動作する仮想86モニタをシステムとして準備することで複数の仮想86タスクの動作を制御することや割り込み処理、I/O

処理の保護を実現していました。

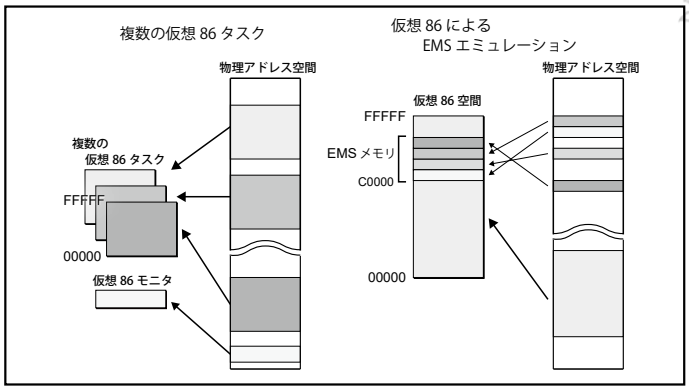
この機能を利用したシステムが1988年に登場するMS-Windows/386です。その前のWindows 2.Xでは、オーバーラップ方式のウィンドウは実現していたのですが、リアルモードで動作するため、DOSアプリケーションは1つしか動作させることができないものでした。それに対して、MS-Windows/386は、仮想86モードを駆使して、複数のDOS窓を開けて、保護機能による安定な動作を実現できたのです。

仮想86モードの保護機能

仮想86は、プロテクトモードのタスクであるため1MBのリアルモード空間内での振る舞いは、リアルモードと同様ですが、その外のメモリ空間へのデータアクセスやプログラム実行では保護機能が働き安定した動作を実現できたのです。

外部へのアクセスは、メモリアクセスのほかに、割り込みとI/O処理があります。複数のDOS環境やシステムからのそれぞれの動作を保障するためには、複雑な制御システムが必要となります。仮想86モードにおける割り込み処理は、プロテクトモードで実行されます。割り込みテーブルではなく、IDTゲートディスクリプタ[注2]を経由するので、リアルモード用の16bit割り込み処理コードを動作させるためには、複雑な前処理が必要になりました。また、I/O入出力処理においては、通常では例外が発生し、例外処理コードが実行され、その中でI/O処理が行われることになります。そこでは、処理時間が大きくなるため、タイミングが重要となるI/O処理では問題が発生することもありました。

▼図1　仮想86モード



実用的な仮想86モードの利用

PC-9801シリーズでは、1989年に日本語MS-DOS 3.3Bが発売され、仮想86モードを使ってプロテクトメモリをソフトウェアEMSメモリとして利用するEMM386.SYSが搭載されました。i80386には、i80286にはなかったページング機能が追加され、4KB単位に物理メモリをリニアアドレス空間に配置できページ単位で保護機能をつけることができました。このページング機能を利用してプロテクトメモリをEMSメモリに割り当てて、高速なソフトウェアEMSメモリを実現したのです(**図1右**)。ハードウェアEMSに比べて16KBサイズのメモリの切り替えには時間を要しましたが、メモリバスに直接つながっているプロテクトメモリのアクセスは高速であり、i80386マシンの優位性が発揮できたのです。

結局、高速なEMSメモリを備えた1つのMS-DOSマシンとしての使い方が、Windowsの普及の前まで続いたのです。

仮想86モードのその後

仮想86モード上のMS-DOSの利用が普及すると、割り込み処理とI/O入出力処理における遅延が問題となり、1993年に登場するPentiumでは仮想モード拡張(VME)が実装され、仮想86モードの実行の効率化を実現したのです。

その後MS-Windowsにおける DOS窓などで仮想86モードは利用されましたが、x86が64bit化されx64コードの実行が可能になると下位互換は32bitコードに移り、16bit互換機能である仮想86モードは、x64モードからは動作できなくなってしまいました。SD

注2) 割り込みと、それに対応する処理を結びつけるテーブルで、ゲートディスクリプタと呼ばれるデータ構造体を介してさまざまな保護機能つきで、割り込みに対応する処理に移行する。



うまくいく チーム開発のツール戦略

第10回 CIツール「Bamboo」で開発プロセスの生産性を上げる

Author リックソフト㈱ 河村 康博



はじめに

前回(2017年8月号)は、バージョン管理システムと課題管理システムを連携させてソフトウェア開発を進めるプロセスを、Bitbucket Server(以下、Bitbucket)とJIRA Software(以下、JIRA)を例に用いて説明しました。バージョン管理システムと課題管理システムは、リックソフトの考える「三種の神器」の第一と第二にあたります。今回はその第三にあたる継続的インテグレーション(CI)システムに着目し、ビルド/テスト/デプロイといった作業の自動化/省力化、さらに「三種の神器」をすべて連携させることで生産性を向上させる方法について紹介します。

CIシステムは、JenkinsやTravis CIなど世の中に多数存在しますが、本稿ではAtlassianが提供する商用WebアプリケーションであるBambooを取り上げます。また、Bambooと連携させるバージョン管理システムと課題管理システムとして、前回に引き続きBitbucketとJIRAを交えて説明します。



CIについて

過去の連載でCIについて触れたこともありますが、本題に入る前に、CIについて再度おさらいをしておきましょう。

CIとは、ソフトウェアのビルド/テストといった作業を継続的に繰り返しながら開発を進めていく開発手法を指します。CIを開発に取り入れることで、コンパイルエラーをはじめとした不具合を早期に発見できるようになります。不具合を早期に発見すれば、あとになって不具合を発見する場合と比べて対応コストが低くなるため、素早く、かつ正しく修正対応が行えます。そのため、CIの導入によって作業効率の向上とソフトウェアの品質の向上が期待できます。

しかし、ソースコードを編集するたびに開発者が手作業でビルドをするのでは、そちらに時間を取られてしまい、むしろ作業効率が悪化します。加えて、繰り返し何度も行う作業を人間が行えば、どれだけ注意していてもいつか必ずミスが発生するものです。

また、単純な手作業でのビルドにはほかにもさまざまなデメリットが存在します。たとえば、特定の環境ではビルド可能でも、ほかの環境ではビルドができないといった環境依存が増えたり、一部の人が「ビルド職人」となり作業が属人化することなどが考えられます。こういった状況を防ぎ、CIのメリットを最大限に活かせるように、ビルドのような単純作業はシステムで自動化するのがお勧めです。



Bambooに作業を任せる

ビルド&テスト

Bambooでは、ビルドの実行プランを「ステージ」「ジョブ」「タスク」という階層構造で設定します。ここが少し複雑に感じる方がいるかもしれませんが、一度覚えてしまえばWebブラウザを使っ



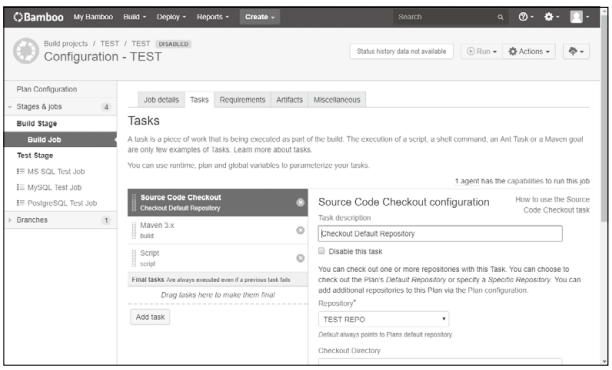
て直感的なUIでビルドの実行プランを設定できるようになるのでぜひ覚えてください(**図1**)。

各概念は大まかに、次のようになっています。

- **タスク**：「ソースコードをチェックアウトする」「Mavem3.xでビルドをする」などの、個別の処理を指す概念です。タスクどうしの並列実行はできず、設定した順番どおりに実行します。
- **ジョブ**：複数のタスクをまとめる概念です。「MySQL環境用のテスト実行」「PostgreSQL環境用のテスト実行」のように、環境ごとに分けて処理を実行したいときに便利です。ジョブどうしは並列に実行可能です。
- **ステージ**：複数のジョブをまとめる概念です。「ビルド用ステージ」「テスト用ステージ」のように、手順を段階ごとに分割するのに用います。ステージどうしの並列実行はできず、設定した順番どおりに実行します。

ビルドの実行プランを作成したら、実行タイミングを設定します。Bitbucketと連携させていれば、編集したソースコードがブランチにコミットされたタイミングでそれを検知し、自動でビルドを実行するように設定できます。また、「毎日深夜3時にビルドを実行する」「開発用ブランチはコミットを検知して自動で、リリース用ブランチは手動でのみビルドを実行する」といったことも可能なので、運用に合わせて最適な実行タイミングを設定できます。

▼図1　ビルドの実行プラン設定



デプロイ

Bambooでは、ビルドやテストだけでなくデプロイの実行も可能です。ビルドの実行プランと同様、デプロイの実行プランを作成し、実行タイミングを設定してBambooにデプロイ作業を任せます。

デプロイの実行タイミングは「ステージング環境に対しては、ある特定のブランチでビルドが成功したら即座に自動でデプロイを実施」「本番リリースの環境に対しては、権限を与えた担当者が確認して手動でデプロイを実施」といったように、環境に合わせて任意に設定できます。手動デプロイの場合でも数クリックの操作だけで簡単かつ確実にデプロイできるので、慎重に行わなければならないデプロイ作業も省力化できます。



簡単に情報を共有できる

ビルドやテストといった作業の自動化や、デプロイの省力化をCIシステムで実現できるようになるだけでも大きなメリットですが、自動化したビルド／テスト／デプロイに関する情報は素早く、適切な形で共有することで真価を発揮します。

Webブラウザを使ってBambooにアクセスすることで情報確認はできますが、「変更を加えたソースコードでビルドが成功したのかどうか」「分岐元のブランチはビルドに成功している安全なブランチなのか」「機能追加したプログラムはどの環境にデプロイ済みなのか」といった情報を見るために、毎回Bambooの画面を見なければならないとなると、開発者やマネージャーにとっては小さな手間が積み重なってしまいます。CIシステムによってもたらされる情報は、普段のルーチンの中で確認できるのが理想的です。

そこで、課題管理システムやバージョン管理システムとCIシステムを連携させ、Bambooにアクセスすることなく情報を確認できるようにしてみましょう。

JIRAで状態を確認

JIRAだけでも進捗管理はできますが、「各課題で対応した内容が問題なくビルドに成功しているのか」「ステージング環境や本番環境にデプロイが行われているのか」といった詳細まではわからず、これらを把握するにはBambooの画面を見たり、各担当者に状況を聞いて回ったりする必要があります。プロジェクトに関わる人員が増えれば増えるほど骨が折れそうですが、JIRAをBambooと連携させることでこの問題は解決します。

前回の記事の中で、ソースコードを保存したときのコミットログ内に、JIRAの課題キー(JIRAが課題をひとつひとつ識別するための番号)を含めることで、JIRAの課題とコミット内容がリンクされ、JIRAとBitbucketのどちらからも相互に情報を参照できることを紹介しました。Bambooも同様に、ビルドの対象となっているコミットのコミットログにJIRAの課題キーが存在していれば自動でJIRAの課題とリンクされ、JIRAとBambooで相互に情報を参照できるようになります。

JIRAの課題の詳細情報を表示する画面には「開発パネル」と呼ばれる領域があり、Bambooで実行されたビルドやデプロイの情報が一目でわかるように表示されます(**図2**)。

また、表示されたリンクをクリックすることで、ビルド情報であれば「実施したテストの件数やそのテストにかかった時間」「ビルドの成果物へのリンク」など(**図3**)、デプロイ情報であれば各デプロイ対象環境について「デプロイが実施済みかどうか」「最後にデプロイを行った日付」など、より詳細な情報も確認できます。

▼図2　JIRAの開発パネル



Bitbucketで事故防止

バージョン管理システムは開発現場になくてはならないものですが、単体では管理されているソースコードの品質が十分であるかどうかは簡単には判断できません。コードレビューを実施していても知らない間に不具合が混入したり、そもそも不具合が混入している状態を起点にして開発を進めてしまう恐れもあります。このような事故が起こると不具合への対応が遅れたり、開発作業自体が非効率になったりして、せっかくCIを開発に取り入れてもそのメリットが失われてしまいます。

そこで、バージョン管理システムをCIシステムと連携させ、よりパワーアップさせましょう。Bitbucketは、Bambooと連携させることで、ビルドが成功しているブランチかどうかをさまざまな画面に視覚的に表示する機能が使用でき、不具合混入の事故を未然に防ぎやすくなります。

たとえば、プルリクエストによりコードレビューを実施する際、レビュー画面内にビルドの結果が表示されます(**図4**)。ビルドに失敗するソースコードであることがレビューを実施する時点で確認できれば、即座にマージを却下できるので、不具合の混入を効率的に防ぐことができます。また、Bitbucketではマージの条件として「ビルドに成功していること」という指定も可能なので、システム的にマージを許可しないことができます。

▼図3　ビルドの詳細情報





▼図4　レビュー画面



ほかにも、ある特定のブランチから分岐させて開発用のブランチを作成する際に、分岐元のブランチがビルドに成功している安全なブランチなのか、それともビルドに失敗している危ないブランチなのかを確認できます(**図5**)。ビルドに成功している安全なブランチであることを前提に開発作業を開始できるので、開発を進めていく中でビルドの失敗が発生すれば、自身の開発作業が原因であることがすぐにわかります。

連携は簡単

Bamboo／JIRA／Bitbucket はすべてAtlassian社の製品なだけあって、システム連携は簡単です。もしそれぞれ単体で利用されている方がいればぜひ連携させてください。各システムのURLを入力してアプリケーションリンクを作成するだけで連携機能を使う準備が整います(**図6**)。



最後に

CIシステム導入による単純作業の自動化／省力化は、昨今求められる生産性の向上には欠かせません。しかし、CIシステムを単体でしか利用していないのでは、いまひとつ「痒い所」に手が届かないと感じる場面があるのも事実です。Bambooを提供しているAtlassian社の製品はシステム間の連携が強みであり、そういった「痒い所」に手が届くのが特徴ですので、この機会にぜひお試しください。

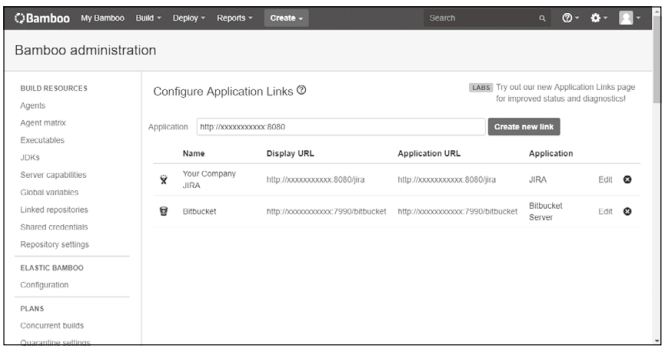
Bambooの機能のより詳細な紹介は、下記過去記事Webサイトの第9回／第10回にありますので、そちらもご覧いただければと思います。SD

※本連載の過去記事は技術評論社のWebサイト「gihyo.jp」でもご覧になれます。
http://gihyo.jp/ad/01/atlassian

▼図5　ブランチの作成



▼図6　URLで簡単に連携



NETGEAR ReadyNAS徹底運用

～大切なデータの保護に耐えうるか実力を試す！～

第3回 ReadyNASのデータ復元

前回はReadyNASで行うバックアップのあれこれを紹介しました。NASの機能を使いこなせばさまざまなバックアップができるため、運用効率が格段に上がることがおわかりいただけたかと思います。そこで、今回はバックアップしたデータをどんなときに復元すればよいかを考えます。いくつかの手法と考え方を紹介しますので、前回と併せてみなさんのReadyNAS運用の参考にしてください。

Author 中山 一弘（なかやま かずひろ）

ロールバックはどう使う?

NASを運用していくうえで、ユーザに与えたフォルダの管理は大切です。ユーザからNAS管理者へはさまざまなニーズが寄せられますが、中には「フォルダ内のデータを紛失した」あるいは「上書きに失敗した」などのクリティカルなトラブルもあるはずです。そのようなケースには個別に対応するしかありませんが、ReadyNASが持ついくつかの機能を使うことで簡単に解決できます。

覚えておきたいのは、フォルダやファイルを以前の状態に戻せる「ロールバック」機能です。広い意味ではデータの「復元」と呼ばれることもありますが、いずれにしてもトラブル発生以前の状態へ戻す作業を指します。

ReadyNASでは、スナップショット（一定のルールで作成したバックアップ）を定期的に取っていれば、そのいずれかの状態にフォルダの内容を含めて戻すことができます。NASでは一般的な機能ですが、ReadyNASでは回数無制限のスナップショットポイントを活用してかなり以前の状態へロールバックすることや、ファイル単位でロールバックすることができます。また、作業もとても簡単で、ちょっとした知識さえあれば誰でも操作が可能な点も見逃せません。

たとえば、PCの入れ替えやトラブルによるOSの再インストール、あるいはドキュメントの紛失や作業中のデータのトラブルなど、ロールバックを活用できるケースはいくつも考えられます。みなさんのビジネスにも応用できますので、ぜひ使いこなしてください。では、実際の方法を解説していきましょう。

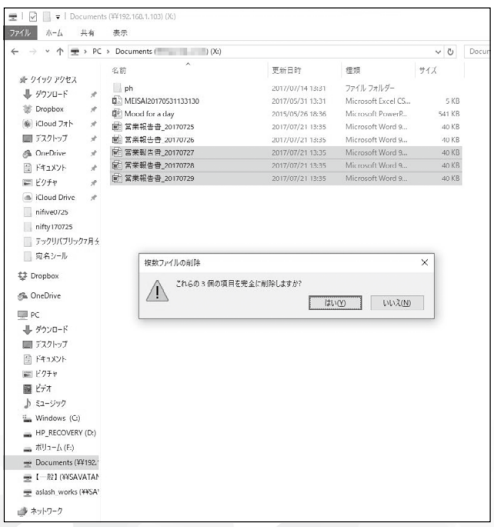
フォルダごと復元する

ReadyNASの標準機能で最も使いやすいスナップショットを例に、フォルダごとにロールバック（復元）する手順を見ていきます。前回（本誌2017年9月号）のスナップショットの取り方と併せて確認してみてください。

まず、共有フォルダにある任意のファイルを誤って削除してしまい、ゴミ箱からの回収も困難だと仮定します（**図1**）。

では復元作業に入ります。まずはReadyNAS OSにアクセスし、「管理者ページ」から問題のフォルダを指定します。フォルダの中身を確認

▼**図1** いつも使っている共有フォルダのドキュメントを誤って削除！

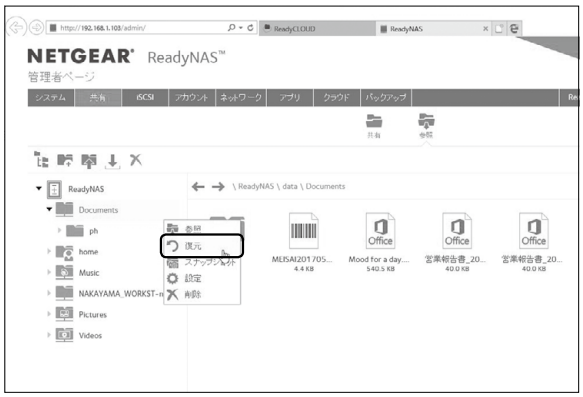


すると、こちらでもやはりファイルがなくなっていることがわかります。フォルダを指定したまま右クリックして、メニューから「復元」を選択しましょう(**図2**)。

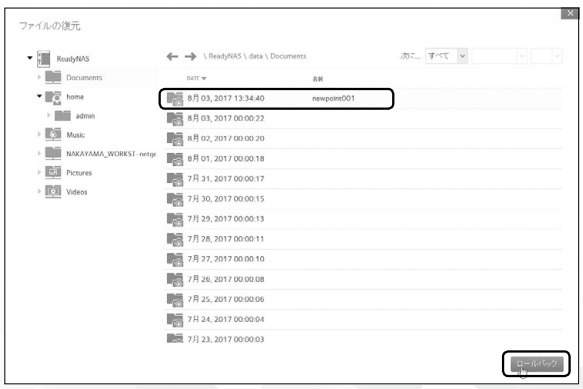
すると「ファイルの復元」が起動するので、リストを確認します(**図3**)。任意の日時のスナップショットに戻れますが、今回はドキュメントを削除してしまった直前に戻れればよいので、最新のスナップショットを指定します。

リストの最上段、つまり最新のスナップショットを指定したら、右下の「ロールバック」ボタンを押します。すると「スナップショットをロールバック」が出てくるので、復元作業の確認のために空欄に「DELETE DATA」と入力します(**図4**)。そのあと、同じウィンドウ内の「ロールバック」ボタンをクリックします。

▼図2 ロールバックしたいフォルダを右クリックし、メニューから「復元」を選択



▼図3 スナップショットのリストを確認。ここでは最上段のスナップショットへロールバックする



復元にかかる時間はファイルの大きさや個数によって変わりますが、しばらく待っていればファイルがスナップショットを取った時点のものに復旧されます。

なお、選択したスナップショットから時系列に沿って新しいものがある場合、それらは失われるので注意してください。つまり、過去に戻った分、その時点から未来のスナップショットは破棄されるということです。破棄されるスナップショットに重要なポイントがある場合は、「クローン」を作って任意の時点のバックアップを取っておくことも可能です(コラム参照)。

ファイル単位で復元する

次に紹介するのは、フォルダ単位ではなく、1つのファイルを復元する方法です。たとえば、ドキュメントを整理しようとして、いくつかのファイルをまとめて削除したが、後日、その中の1つは必要だったことが発覚した、といったときに便利な機能です。フォルダ単位のロールバックに比べて復元時間が圧倒的に短く、作業負担も軽減できます。では、さっそく説明しましょう。

▼図4 「スナップショットをロールバック」の空欄に「DELETE DATA」と入力。「ロールバック」ボタンを押して復元作業を決定する



COLUMN

スナップショットのクローン

ReadyNASのスナップショットには「クローン」と呼ばれる機能があります。簡単にいうと、スナップショットの時点でのイメージをそのまま任意のフォルダとして複製できるのです。通常のロールバックは指定のフォルダを上書きするのに対して、クローンでは別フォルダとして復元されます。当然ながらフォルダが増えますから、NASの容量もその分使用されますが、何らかの理由でスナップショットを破棄する必要があるにもかかわらず、特定のポイントの複製だけは確保しておきたいという場合には便利でしょう。知っておいて損のないしくみなので、ぜひ覚えておいてください。

▼図A　クローンの作成方法



任意のスナップショットを指定し、メニューから「クローン」を選択

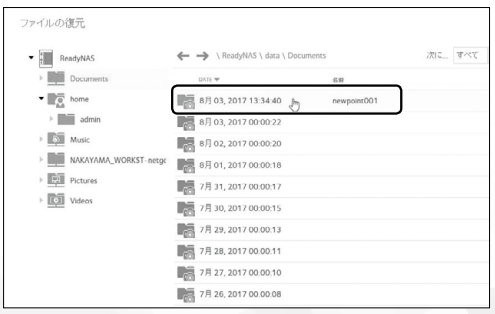
クローンの名前を付ける。これが複製されるフォルダ名となる

しばらく待てばフォルダの複製が作られる。もちろん、中身もスナップショットの時点のものが復元される

まずは先ほどと同様、共有フォルダにあるいくつかのドキュメントをまとめて削除したと仮定します。今回はその中から最新版の「営業報告書_20170729.doc」というファイルを復元してみます。

ReadyNAS OSの「管理者ページ」から任意の共有フォルダを指定し、右クリックメニューから「復元」をクリックします。今回も1つ前のスナップショットに戻ることから始めます。ここではリストの最上段にあるスナップショットをダブルクリックしてください(**図5**)。

▼図5　リストの最上段、ドキュメントを削除してしまう直前のスナップショットをダブルクリックする



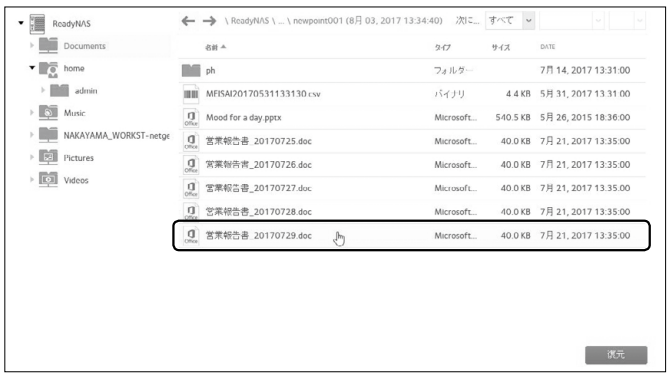
すると、スナップショットで保存したフォルダの内容が表示されます。この時点ではどのファイルやサブフォルダも復元されておらず、単純にリストとして表現されているだけです。この中から今回は復元したいファイル「営業報告書_20170729.doc」だけを選択し、右下の「復元」ボタンを押します(**図6**)。そのあと、いくつかのダイアログが出ますので指示に従って進めていきましょう(**図7**)。

これで復元したかったファイルのみを救出することに成功しました。この機能を応用すると、もう少しユニークな使い方もできます。たとえば、作業中のドキュメントの更新や上書きに失敗したり、うっかり大事な記述を削除してしまったりしたときにも、この機能で復旧させることができるのです。

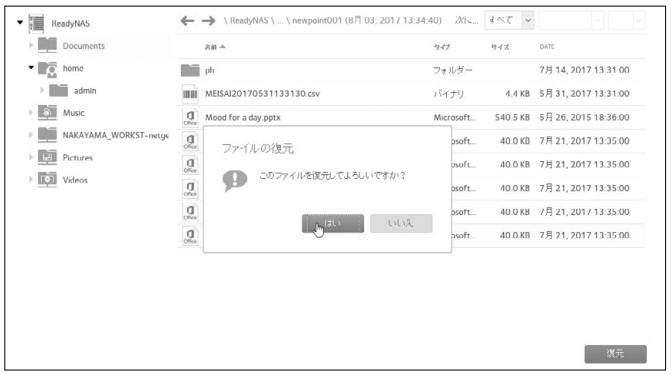
これには、作業の更新に合わせたタイミングでスナップショットを取っていることが前提となりますが、ファイル単位というよりも作業フロー単位でも任意の地点までロールバックできることがおわかりいただけると思います。も

ちろん業務内容にもよるでしょうが、こうした機能を使いこなすことで、安全なドキュメント運用のヒントになるはずです。

▼図6　スナップショットを開き、その時点のファイルやサブフォルダのリストから復元したいファイルを指定



▼図7　ダイアログによる指示に従って進める



スナップショット機能だけでもバックアップが充実

2回にわたってスナップショットを中心にバックアップや復元の方法を解説しました。ReadyNASならこれだけの機能を実に簡単に使いこなせることがおわかりいただけたかと思います。

安心を担保するためのバックアップから、業務のスムーズな進行をサポートできる復元方法まで、スナップショット機能を使いこなすだけで幅広いバックアップ&ロールバックが可能になります。これらの機能を知っていれば、あとは業務内容に合わせてルールを作っていくだけで、これまでと比べてデータ運用の効率を確実にステップアップできます。みなさんもぜひReadyNASシリーズのスナップショットを活用して、効果的なNAS運用に役立ててください。それではまた！SD

COLUMN

スナップショットはどこに格納されている？

ここまでスナップショットを理解してくると、スナップショットのファイルがどこに保存されているのか気になるかと思います。結論から言うと、スナップショットはそのフォルダ内に隠しフォルダのような状態で、通常ではアクセスできない環境として保存されています。

表示方法は簡単で、ReadyNAS OSの「管理者ページ」から任意のフォルダを指定し、右クリックメニューにある「設定」の「スナップショット」タブを開き、「スナップショットへのアクセスを許可する」にチェックを入れるだけです（**図B**）。こうすると、スナップショットが格納されているフォルダ「snapshot」が表示されるようになるので、内容も確認できるようになります。

▼図B　任意のフォルダの右クリックメニューから「設定」を選択。出てきたウィンドウから「スナップショット」タブを開き、「スナップショットへのアクセスを許可する」にチェックを入れる



Retty Tech Night 2017＠Rettyオフィス

日本最大級の実名グルメサービスRetty（**https://retty.me**）を運営するRetty㈱は、3,000万人の月間利用者数を誇る事業へと発展した。現在はアジアを中心に海外向けサービスも開始し、その急成長ぶりは2010年の創業以来6度目のオフィス移転ということからも見て取れる。そうした中、ITエンジニアに対するコミットもイベント開催の形で継続的に行われており、これまでにRetty Tech Cafeとして8回行っている。9回目は麻布十番への新オフィス移転を記念し、「Retty Tech Night」とイベント名を変えての開催となった。そのテーマは「レッティ社員のレッティ社員によるレッティに興味のある方のためのイベント」。ソフトウェア開発とサーバ運用、そして機械学習と、さまざまな面からRettyの試みを紹介する充実した内容となった。イベント終了後の懇親会ではグルメサービスの本領を生かし、有名レストランからのケータリング料理が参加者にふるまわれた。講演内容は次のとおり。

・KotlinとReduxをAndroidアプリに導入したら（**写真1**）
・バックエンドが異なるサービスの差分をAPIで吸収してみた話（**写真2**）
・Rettyのインフラを支える技術（**写真3**）
・Rettyの機械学習のこれまでまとめ（**写真4**）


▲写真1　福井 敦子氏（ソフトウェアエンジニア）


▲写真2　石田 憲幸氏（ソフトウェアエンジニア）


▲写真3　鈴木 巧氏（SRE）


▲写真4　氏原 淳志氏（ソフトウェアエンジニア）

CONTACT
Retty㈱　URL https://corp.retty.me

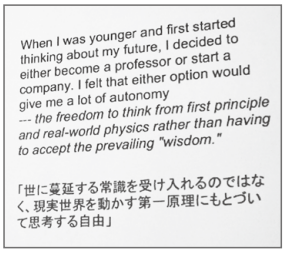
ITインフラエンジニアのためのイベント「July Tech Festa 2017」基調講演、招待講演レポート

8月27日、産業技術大学院大学にて「July Tech Festa 2017」が開催され、30のセッションと2コマのハンズオンが行われた。基調講演と招待講演をレポートする。

基調講演

Google Cloud、ソリューションアーキテクトの中井悦司氏が登壇。本イベントのテーマである「ITエンジニアの本質を極める」にあわせ、講演テーマは「『ITエンジニアリングの本質』を考える」。そこで引用したのはGoogle創業者の1人、ラリー・ペイジの言葉だ（**写真1**）。

この言葉を実践している最たるものとして紹介したのがGoogleのデータセンターネットワーク。12年前、分散ネットワークシステムを構築する際にネックとなっていた帯域の分断化を解決し、より高速でより信頼性の高いものを目指して改良を重ねてきた経緯を紹介。その時々の技術や情勢に応じて、要求されるものの最適解を毎回ゼロから考え直すというのがGoogleのエンジニアリングであり、そこから数々のイノベーションが生まれている、といった話がなされた。中井氏の講演に使われた資料はSlideshare（**https://www.slideshare.net/enakai/it-78447313**）で公開されている。



▲写真1　ラリー・ペイジの言葉

招待講演

登壇したのは、日本マイクロソフト㈱ 執行役員 最高技術責任者の榊原彰氏。「日本のエンジニアが今取り組むべきこと」と題した講演を行った。榊原氏がかつて子どもたち向けに行った「エンジニアってすばらしい！」という講演のエピソードにはじまり、社会にあるさまざまな課題を解決するために、データの「コンテクスト」を知り、社会をモデル化し、製品・サービスの市場投入サイクルを短くすることの重要性を説いた。そしてなにより問題を解決に導ける“エンジニア”の可能性を語り、「わずかな時間でも社会に関心を持ち続け、自分にできることを考える。それを今日から少しずつでいいので実践していきましょう」と聴講者にメッセージを贈って締めくくった。

CONTACT
July Tech Festa 2017　URL https://2017.techfesta.jp

Event

10月21日、22日
openSUSE.Asia Summit 2017 Tokyoを開催

10月21日と22日、電気通信大学（東京都調布市）において「openSUSE.Asia Summit 2017」が開催される。

「openSUSE」はサーバ、デスクトップなど多目的な用途に使えるLinuxディストリビューション。SUSE社の製品「SUSE Linux Enterprise」のベースにもなっている。

openSUSE.Asia Summitは、アジア地域のopenSUSEコミュニティ（貢献者、ユーザ）が一堂に集まり、交流し、楽しむイベント。普段はオンラインで活動しているコミュニティのメンバーが、実際に対面し、その声音を聞きながら、それぞれの知識や経験を共有し、openSUSEを中心とした多彩なFLOSS（Free/Libre and Open Source Software）に関する技術を学ぶことができる。今回は口頭発表とハンズオンセミナーが予定されており、発表ではopenSUSEのほかDocker、Ansible、Libre Office、Linuxパフォーマンス分析に関するものも行われる。

●開催概要

日程	10月21日（土）～10月22日（日）
場所	電気通信大学（東京都調布市調布ヶ丘1-5-1）
参加費	無料
参加方法	事前登録制（下記Webサイトで随時案内する予定）

CONTACT
openSUSE.Asia Summit 2017
URL https://events.opensuse.org/conference/summitasia17

Software

パラレルス、
「Parallels Desktop 13 for Mac」発売

パラレルス㈱は8月25日、Mac向けの仮想化ソフト「Parallels Desktop for Mac」の最新バージョン「Parallels Desktop 13 for Mac」を販売開始した。

Parallels Desktopは、Mac上でWindowsやほかのmacOSの仮想マシンを実行できるソフト。最新バージョンでは、次の機能強化が図られた。

・Mac Touch Bar
Officeや各種ブラウザといったWindowsアプリケーションのボタンをMac Touch Barに表示可能に。未対応のアプリについては、ユーザがカスタマイズできるしくみも提供

・Windows People Bar
今後Windowsに実装予定の、タスクバーに連絡先を顔写真のアイコンとして表示できる「People Bar」が、Mac上のDockでも利用可能に

価格は、通常版が8,500円（税込）、アップグレード版が5,300円（税込）。Visual Studio、Vagrant、Chef、Jenkinsなどの開発ツールもサポートする「Pro Edition」は、1年期間更新版で10,000円（税込）となっている。

CONTACT
パラレルス㈱ URL http://www.parallels.com/jp

Software

グレープシティ、
「SpreadJS」を9月27日に発売

グレープシティ㈱は、WebアプリにExcelライクなUIとグリッドUIを実現するJavaScript製品「SpreadJS」を9月27日に発売する。

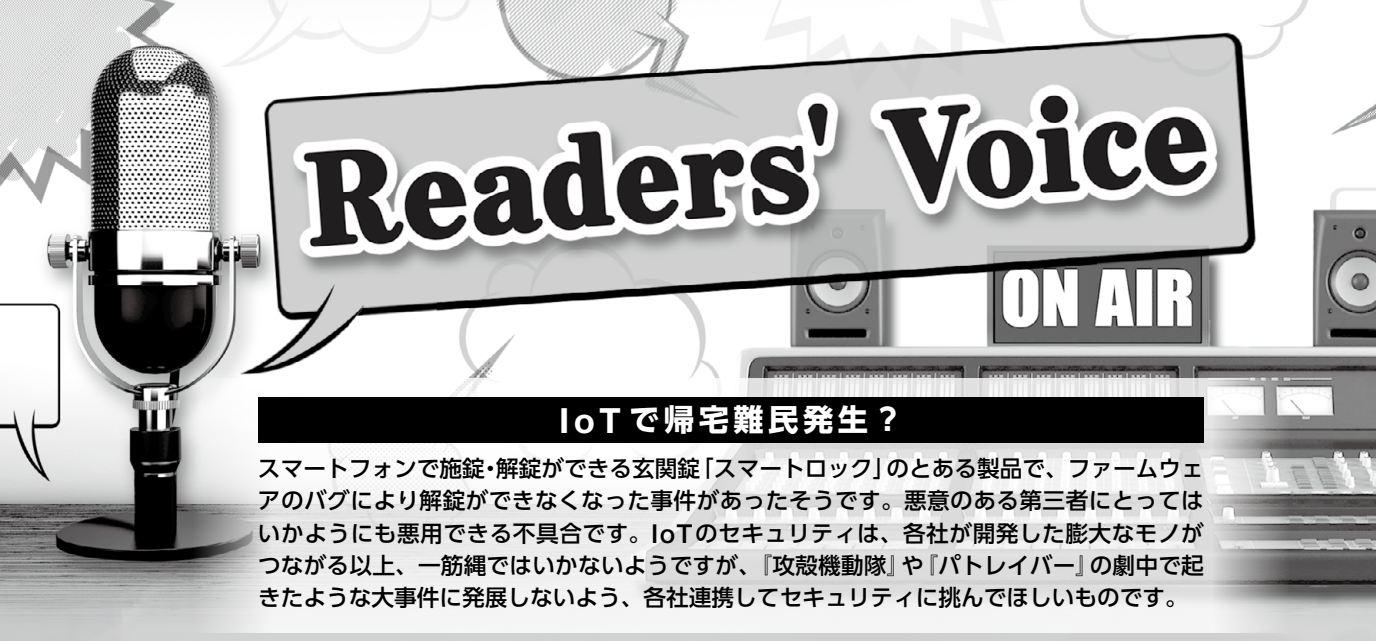
SpreadJSは、Webアプリのデータ表示に特化したHTML5 スプレッドシートコンポーネント。Canvasベースで必要な機能ファイルのみをロードするので、軽量かつハイパフォーマンスなWebアプリを開発できる。次の2つのUIコントロールを収録する。

・Spread.Sheets
ExcelライクなUIを実現できる。表計算関数やフィルタリングをはじめとした豊富なExcel互換機能を搭載

・Spread.Views
グリッドUIを実現できる。標準的なグリッド表示に加え、カレンダーやガントチャートなどのデータ表現を持つ

対応ブラウザはInternet Explorer 9以上、Microsoft Edge、Chrome、Firefox、Safari 5.1以上。価格は、1開発ライセンスが150,000 円（税別）、配布ライセンスが600,000円（税別）。

CONTACT
グレープシティ㈱ URL http://www.grapecity.com



IoTで帰宅難民発生？

スマートフォンで施錠・解錠ができる玄関錠「スマートロック」のとある製品で、ファームウェアのバグにより解錠ができなくなった事件があったそうです。悪意のある第三者にとってはいかようにも悪用できる不具合です。IoTのセキュリティは、各社が開発した膨大なモノがつながる以上、一筋縄ではいかないようですが、『攻殻機動隊』や『パトレイバー』の劇中で起きたような大事件に発展しないよう、各社連携してセキュリティに挑んでほしいものです。

2017年8月号について、たくさんの声が届きました。

第1特集　私も機械学習エンジニアになりたい！

機械学習エンジニアのスタートラインとして、機械学習の全体像、ディープラーニングの基本的な考え方、GPU搭載の機械学習用PCの作り方、現場で求められている人材や知識について解説しました。

機械学習についてのノウハウというよりも、機械学習がどういうものかという視点からの解説により、機械学習に対する理解が深まりました。また、実際の機械学習エンジニアの実体験なども記載されていて、具体的な内容で勉強になりました。

r_tihiroさん／東京都

難しいですが、今後いろいろな分野で活用されていくのは確実そうなので、少しずつ理解を深めていきたいと思います。　檜山さん／埼玉県

まさに、どこから手を付けて良いかわからない状態だったので、ちょうど良かった。　ほまれさん／千葉県

今流行りの機械学習とは何か、また、その使い方や応用事例を学ぶことができて良かった。　りょうじさん／東京都

ディープラーニングやCNNなど、言葉ばかりはよく聞きますが中身がさっぱりだったこの分野、今回の特集でだいぶわかった気になれました。

penpenさん／神奈川県

第2章「ディープラーニング入門」を呼んで、Character-level CNNについて初めて知りました。処理から応用例まで具体的にわかりやすく書かれており、たいへん参考になりました。

卯木さん／千葉県

流行っているのはわかるけれど、どこから手を付ければ良いのか、そもそも自分には関係ないのでは、と考えていた読者が多いようでした。本特集で、身近に感じてもらえるようになった読者が増えた印象でした。

第2特集　エンジニアのためのうけるプレゼン・すべるプレゼン

IT系勉強会ブームの中、エンジニアに対して改めて、発表する力＝プレゼン力が問われ始めています。プレゼンの達人が、「うけるプレゼン」「すべるプレゼン」を示しながら、プレゼンカアップのポイントを解説しました。

つい多くを語り過ぎるプレゼンですが、たいへん参考になりました。

QKobさん／富山県

会社でプレゼンをすることがあるので、とても参考になりました。

yoshitakaさん／神奈川県

人前で話すのは苦手なので、共感できた。　増田さん／千葉県

これすごくおもしろかった！　いきなり登壇はレベルが高いので、社内での発表に活かそうと思う。

杉澤さん／神奈川県

お客様へのプレゼンや構築中の打ち合わせなど、エンジニアが話す場は結構あるのですが、自分も含め苦手な人がすごく多いので、この特集を参考にして場数をこなしていきたいです。

にわとりさん／東京都

伝える大事さを意識することは、エンジニアがキャリアを積むには欠かせないことです。プレゼンの肝心なことを取り上げて、もっと拡散していけたら良いと思います。　E・Rさん／東京都

プレゼンテーションは大学の研究生時代から苦手でした。やはり、場を設け

8月号のプレゼント当選者は、次の皆さまです

①SSD「S300TL240K」
對馬剛志様（埼玉県）

②SIMPLE SMART BOX
大山嘉治様（埼玉県）

③Octocat ぬいぐるみ
宮本紘行様（神奈川県）

④『人狼知能で学ぶAIプログラミング』
ひで魚様（東京都）、三美昌之様（東京都）

⑤『動かして学ぶセキュリティ入門講座』
ふぃれ様（静岡県）、吾郷洋平様（茨城県）

⑥『プログラマのためのGoogle Cloud Platform入門』
青山豊様（富山県）、柿原研様（東京都）

⑦『Webフロントエンドハイパフォーマンスチューニング』
ぴょうへい様（大阪府）、島津真人様（東京都）

※当選しているにもかかわらず、本誌発売日から1ヵ月経ってもプレゼントが届かない場合は、編集部（sd@gihyo.co.jp）までご連絡ください。アカウント登録の不備（本名が記入されていない場合、プレゼント応募後に住所が変更されている場合など）によって、お届けできないことがあります。2ヵ月以上、連絡がとれない場合は、再抽選させていただくことがあります。

て試行錯誤していくしかありませんね。 とっぽさん／愛知県

スティーブ・ジョブスのスピーチはすばらしいお手本だと考えています。 鈴木さん／熊本県

仕事上避けられない、苦手なので克服したい、など、プレゼンのコツには需要があるようでした。勉強会やカンファレンスでの登壇は、ステップアップとしての転職にもつながる場合があるそうで、プレゼン力はぜひ鍛えていきたいですね。

一般記事　「Mastodon」旋風からわかるSNSの未来

Twitterの代替として注目を浴びた分散型SNS「Mastodon」について、それを支える技術を解説しながら、今後SNSがどのような行末をたどるのかを考察しました。

最近ちらほら聞くようになりましたね。読んでみて自分でも運用してみたいなとか思いました。 クラウドさん／京都府

気にはなれども、手は出していなかったため、ありがたい記事です。 ゆめかけさん／神奈川県

SNSはLINEさえも使っていないが、中央管理ではない点には興味があります。 かずさん／千葉県

インスタンスが独自の進化をしていく開発スタイルには驚きました。 あいくたんさん／神奈川県

Mastodonは使っていないが、SNSの今後を考えるうえで参考になった。 牧田牧場さん／東京都

大きなブームになったMastodon、ほかのSNSにはない分散型という点に興味を持った読者が多いようです。いろいろと使い所を検討されている方も多いようで、社内SNSとして構築するのもおもしろそうですね。

短期連載　人工知能時代のLispのススメ【2】

現在使われる多くのプログラミング言語に影響を与えた「Lisp」について、その画期的なしくみや思想に、あらためて注目する短期連載です。第2回は、関数型プログラミング・人工知能・オブジェクト指向とLispとの関わりについて解説しました。

オブジェクト指向には興味があります。Lisp、なんか懐かしいです。 風のピエロさん／長野県

以前Lispを学習していたので復習の観点で読みました。来月も楽しみです。 チャチャ丸さん／東京都

昔にEmacsやxyzzyエディタで少しかじった程度だったので、懐かしい気持ちで読めた。 山添さん／東京都

Lispでまとめられた記事はあまり見たことがなく、参考になりました。 とーふやさん／神奈川県

歴史があり、かつ関数型やオブジェクト指向といった多様な側面を持つLispですので、温故知新的な声が多く寄せられました。昔を懐かしんでいる方もいる一方、新鮮味を感じている方もいるようでした。

コメントを掲載させていただいた読者の方には、1,000円分のQUOカードをお送りしております。コメントは、本誌サイト**http://sd.gihyo.jp/**の「読者アンケートと資料請求」からアクセスできるアンケートにてご投稿ください。みなさまのご意見・ご感想をお待ちしています。

次号予告

Software Design
November 2017

2017年11月号
定価（本体1,220円+税）
184ページ

10月18日発売

［第1特集］現場で培われた実践ノウハウ

データ分析に効くSQL50本ノック

SQL初学者のエンジニアも営業担当も
自分でデータ分析ができるようになる!

［第2特集］そう言われてみればうまく説明できない

なぜ、コンピュータは割り算が下手なのか?

CPUの中の動きからわかる、効率のよいプログラミングの手がかり

［特別企画］

・エンジニアの負担を軽減したRedashのデータ可視化ノウハウ
・ホワイトボックススイッチって何?

※特集・記事内容は、予告なく変更される場合があります。あらかじめご容赦ください。

お詫びと訂正

以下の記事に誤りがございました。読者のみなさま、および関係者の方々にご迷惑をおかけしたことをお詫び申し上げます。

■2017年9月号
●P.120　連載「シェル芸人からの挑戦状」、図2-4
［誤］$ sed -nr ':a s/^/@/;p;/^.{256}/!t a' <<<'' | sed -n '=;s/.*/touch &; rm &/e' 2> /dev/null | tail -n 2 | head -n 1
［正］$ sed -nr ':a s/^/@/;p;/^.{300}/!t a' <<<'' | sed -n 's/.*/touch &; rm &/e' 2>&1 | grep touch | head -n 1 | tr -dc @ | wc -c | xargs -I{} expr {} - 1

SD Staff Room

●5歳になる娘の爆発的な成長がめざましい。日に日に難しい言葉を覚え、ひらがなが読めるようになったあとは、自分からカタカナにチャレンジしている。古いDSがあってまだ使えたので「美文字トレーニング」ソフトを与えてみた。試行錯誤しながら遊ぶ娘を見て、「デジタルネイティブだ」と思わず口に出た。（本）

●お盆休みに、期限が切れてしまったパスポートの更新に行った。この期間は一年中で一番混雑する時期らしく、番号札は130番後。結局3時間かかったが無事申請できた。しかし申請する人々の準備の悪さは想像以上。1人10分以上はザラだ。書き直しさせられる人、ゴネる人多数。楷書って何?ってマジか。（2分済幕）

●夏休みに長野へ行ってきました。レンタカーで久しぶりに山道を走行。めちゃくちゃ緊張して運転でヘトヘトに。でもクルマのおかげで山の中にある、忍者好きにはたまらない戸隠村にも寄れました。戸隠民俗館にある「忍者からくり屋敷」は、子どもっぽいかと思っていましたが予想以上に楽しめました!（キ）

●街でズボンをはき忘れてトランクスだけで歩いている人を見かけました。が、実際はトランクスに見える短パンなのでしょう。でも、事実がどうであろうと他人にトランクスに見えている時点で、その服装は失敗なのではないかと思いました。編集・校正も常に第三者の目線で冷静に行わないと!（よし）

●大好きな作品『デジモンテイマーズ』のBlu-rayボックス化が決定しました。DVD版にはプレミアム価格が付いていたので手が出ず、今回やっと手元に置けます。今になって思えばなかなか設定が凝っているアニメで、劇中にはカーネルやオープンソースプロジェクトといった用語も登場します。（な）

本誌に記載の商品名などは、一般に各メーカーの登録商標または商標です。　© 2017 技術評論社

ご案内
編集部へのニュースリリース、各種案内などがございましたら、下記宛までお送りくださいますようお願いいたします。

Software Design 編集部
ニュース担当係

［E-mail］
sd@gihyo.co.jp

［FAX］
03-3513-6179

※FAX番号は変更される可能性もありますので、ご確認のうえご利用ください。

Software Design
2017年10月号

発行日
2017年9月18日

●発行人
片岡　巌

●編集人
池本公平

●編集
金田富士男
菊池　猛
吉岡高弘
中田瑛人

●編集アシスタント
根岸真理子

●広告
松井竜馬
大橋　涼
目良直子

●発行所
㈱技術評論社
編集部
TEL：03-3513-6170
販売促進部
TEL：03-3513-6150
法人営業課（広告）
TEL：03-3513-6165

●印刷
図書印刷㈱



この個所は、雑誌発行時には広告が掲載されていました。編集の都合上、
総集編では収録致しません。



この個所は、雑誌発行時には広告が掲載されていました。編集の都合上、
総集編では収録致しません。