

# ブロックチェーンによるゲーム内乱数の 信憑性確認法の提案

プロジェクトマネジメントコース  
ソフトウェア開発管理グループ  
矢吹研究室  
1442020  
大木崇雅

# 目次

第 1 章	序論	2
第 2 章	背景	3
第 3 章	目的	4
第 4 章	手法	5
4.1	Chocolatey について . . . . .	6
4.2	VirtualBox について . . . . .	11
4.3	Vagrant について . . . . .	16
4.4	docker について . . . . .	25
4.5	docker compose について . . . . .	28
4.6	naivechain について . . . . .	31
4.7	Bitcoin Core について . . . . .	39
第 5 章	結果	42
5.1	naivechain による P2P ネットワーク . . . . .	43
5.2	Bitcoin Core . . . . .	47
第 6 章	考察	59
第 7 章	結論	60
	参考文献	61
	謝辞	62

# 第 1 章

## 序論

ビットコインを始めとする仮想通貨の存在が広く知られるようになってから，ブロックチェーン技術にも注目が集まっている．ブロックチェーンとは分散型のコンピューターネットワークであり，データベースを中央に置かずに分散して取引記録を管理している [1]．

ブロックチェーンは 2008 年にサトシ・ナカモトを名乗る人物が基本理論を提唱したビットコインを実現するための技術である．記録データを「ブロック」と呼ぶ小分けしたデータに加工し，順番に関連付けして鎖 (チェーン) のように連なる構造を取る．ブロックが連なる同一の記録データを複数のコンピューターが管理・保存するというもので，コンピューター同士がブロックを比べてデータを更新する．

## 第 2 章

### 背景

2017 年 11 月 15 日に株式会社 Akatsuki が提供しているソーシャルゲームで有料アイテム抽選装置の確率の不正が疑われ、会社の時価総額が暴落した事件があった。このような事件をデータの改ざんが困難であるブロックチェーン技術を用いて解決できるのではないかと考えた。ブロックチェーンは利用者がそれぞれ同じデータを保有することで、単一のシステムや管理組織に依存しない新たなシステム基盤技術である。

データの改ざんが困難な理由は 2 つある。1 つはあるコンピュータ上に存在するブロックを不正に書き換えても、他のコンピュータ上の記録と異なるブロックを多数決で判断して排除する為だ。全体の 50 % 以上のコンピュータ上の記録を書き換えないと改ざんできない仕組みである [2]。もう 1 つの理由は常に新しいブロックが増え続けるからだ。新たなブロックが生成される速度を上回る速度でブロックを書き換える計算能力を持ったコンピュータがなければブロックを改ざんする事は不可能である。

ブロックチェーンの応用分野は仮想通貨などの金融サービス業に限らず、「改ざんできないデータを共有する」メリットがある業務は対象になり得る。本研究ではブロックチェーンの、データの改ざんが困難であるという特徴に重点を置いて研究を進める。

## 第 3 章

### 目的

ソーシャルゲームの有料アイテム抽選装置での抽選結果をブロックに書き込んでユーザー間で共有・閲覧できるようにすることが本研究の目的である．今回は有料アイテム抽選装置をサイコロで代替し，サイコロの出目を複数のノード間で共有・閲覧可能な環境を再現する．サイコロの出目が記録されたそれぞれのブロックからデータを取り出し，集計して乱数に偏りがいないか調査する．

## 第 4 章

# 手法

疑似乱数列生成器の 1 つであるセルメンヌツイスタを用いてサイコロの疑似乱数を発生させ、乱数データを CSV ファイルに保存するプログラムを作成する。乱数データの入ったブロックを P2P ネットワークで共有できる `naivechain` のブロックチェーン上に追加する。

## 4.1 Chocolatey について

### 4.1.1 Chocolatey とは

Chocolatey は Windows のためのパッケージ管理ツールである。Chocolatey を使うことによって Windows 上で動作するソフトウェアをコマンドラインからインストール、アンインストール、アップデート、検索を行うことができる。naivechain を使用するために Chocolatey を用いて環境を構築する。基本的には Chocolatey と公式インストーラーによる導入の 2 通りのインストール方法を解説する。

## 4.1.2 Chocolatey のインストール

Windows のコマンドプロンプトで管理者として実行する方法.

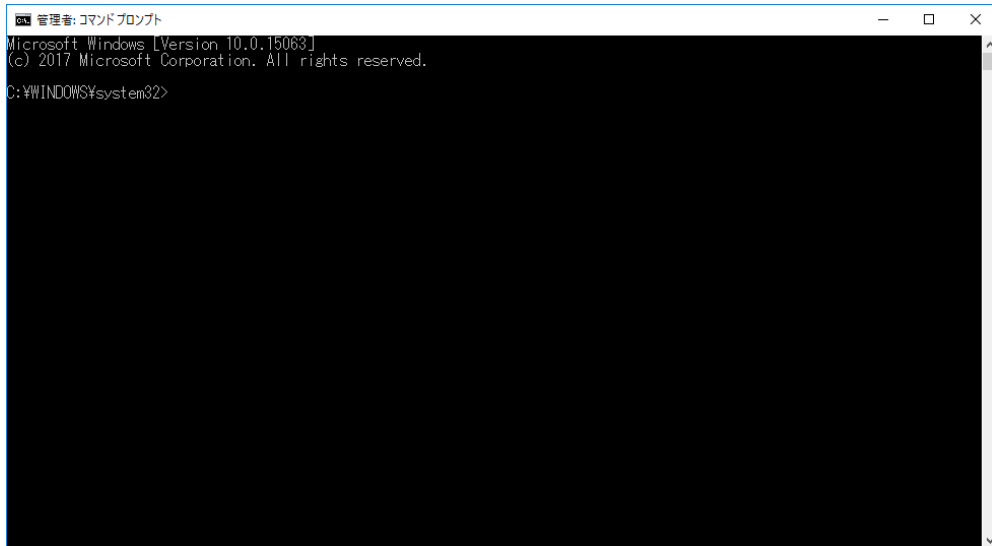


図 4.1 管理者で起動したコマンドプロンプト

この画面から以下のコマンドを入力する.

```
@powershell -NoProfile -ExecutionPolicy Bypass -Command "iex ((New-Object  
System.Net.WebClient).DownloadString('https://chocolatey.org/install.ps1'))  
" && SET "PATH=%PATH%;%ALLUSERSPROFILE%\chocolatey\bin"
```

インストール終了後コマンドプロンプトを再起動して

`clist`

と入力して, ソフトウェア一覧が表示されればインストール成功である.



Chocolatey で使用できるコマンドは以下の通りである。なお、すべて省略されたコマンドである。

表 4.1 Chocolatey で使用できるコマンド

cinst [パッケージ名]	指定したパッケージをインストールする。インストールできるパッケージは Chocolatey の公式サイトから確認できる。
cup [パッケージ名]	指定したパッケージをアップデートする。all を指定すると全パッケージをアップデートする。逆に指定しなければ Chocolatey 本体をアップデートする。
cuninst [パッケージ名]	指定したパッケージをアンインストールする。
clist	Choco でインストールできるパッケージをすべて出力する。
clist -lo [パッケージ名]	指定したインストール済みのパッケージを出力する。パッケージ名を指定しなければすべてのパッケージを対象にする。

以下はホームページからインストールやパッケージの検索を行う方法. google 検索で URL 欄に `https://chocolatey.org/` と入力して chocolatey のサイトを表示し, Install Chocolatey Now をクリックする.

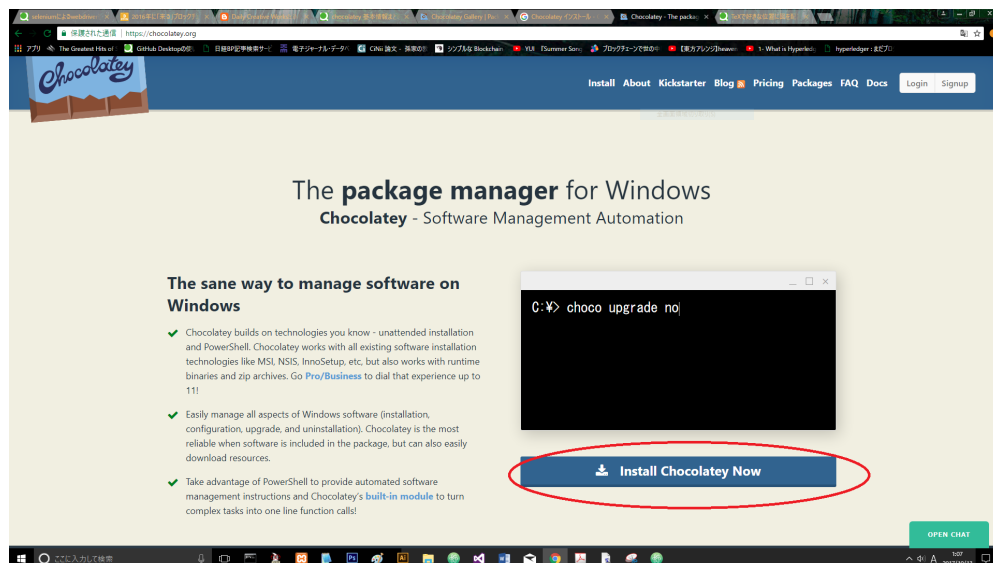


図 4.2 Chocolatey ダウンロードサイト

その後画像の more options を選択. その後, インストールに使うアプリケーションに応じたコマンドをコピーする. パッケージページに飛ぶと, Choco でインストールできるパッケージを検索し, インストールのコマンドを知ることもできる.

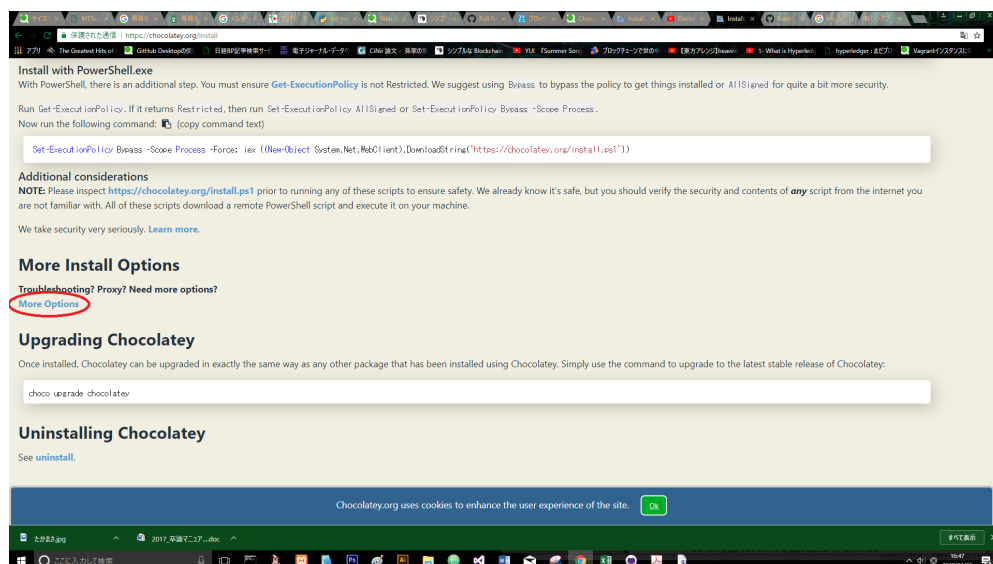


図 4.3 Chocolatey ダウンロードサイト

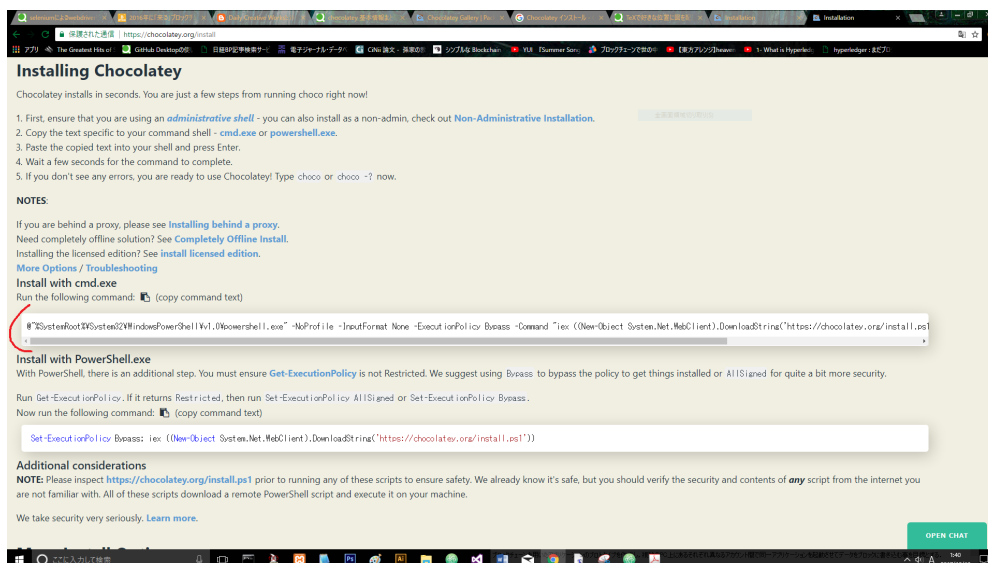


図 4.4 Chocolatey にあるコマンドコード

Install with cmd.exe に書かれているコマンドをコピーする。

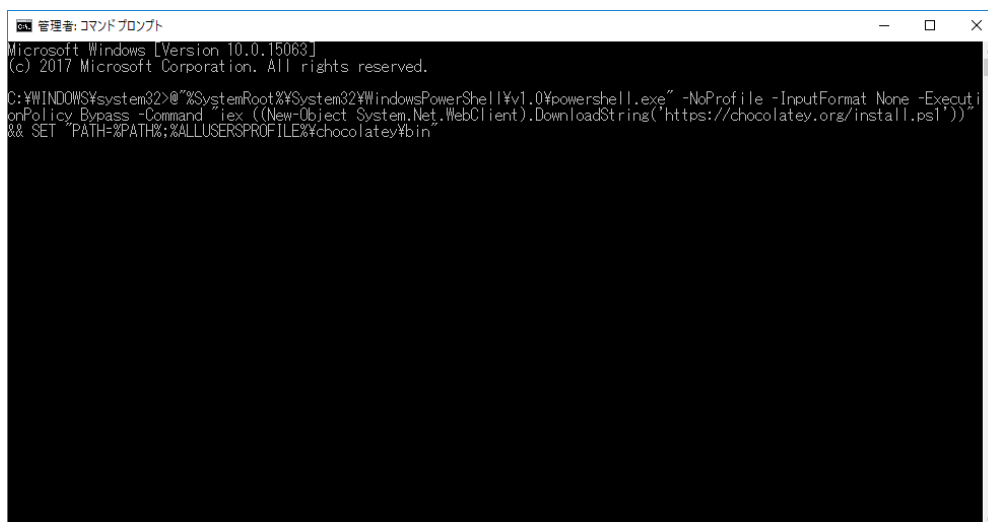


図 4.5 管理者権限のコマンドプロンプト

起動した管理者権限のコマンドプロンプトにコピーしたコードをペーストして実行する。

## 4.2 VirtualBox について

### 4.2.1 VirtualBox とは

VirtualBox は，使用している PC 上に仮想の PC を作成し，別の OS をインストール・実行できるフリーの仮想化ソフトである．

VirtualBox はコンピューター上で直接動作している通常の OS にとってはアプリケーションソフトの一つであり，他のソフトと同じように起動することができる．起動すると仮想的なコンピューターが構築され，元の OS とは独立に別の OS を起動することができる．VirtualBox が実行されている OS をホスト OS，VirtualBox 上で実行されている OS をゲスト OS という．

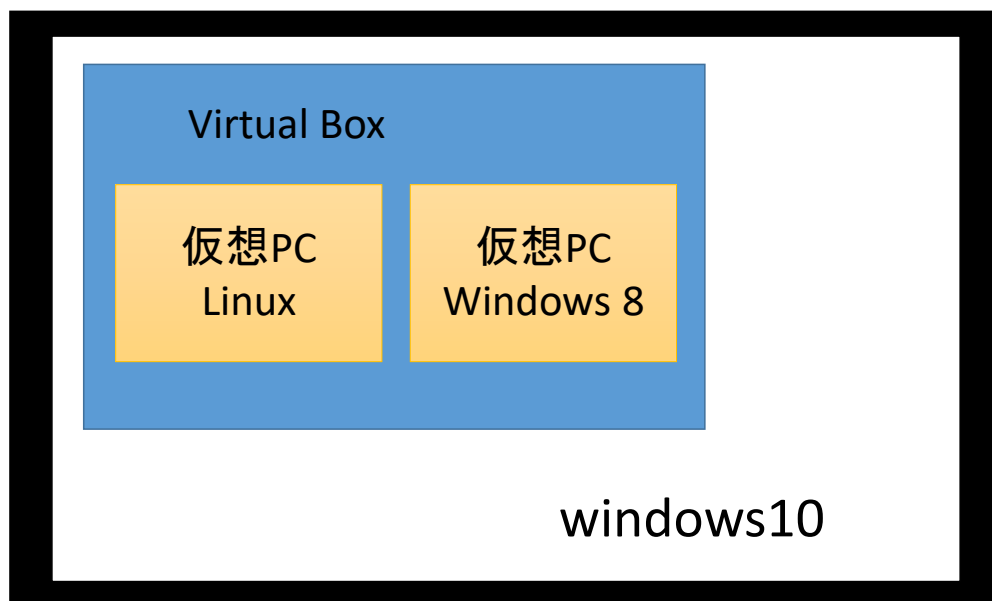


図 4.6 仮想マシンのイメージ

元は独立系のソフトウェア企業が開発・販売していた製品だったが，開発元が Sun Microsystems 社に買収されたため，Oracle 社が開発元となり，正式名称も「Oracle VM VirtualBox」となった．また，VirtualBox 本体は GPL に基づいてオープンソースソフトウェアとして公開され，誰でも自由に入手・利用・改変・再配布などが行える．同社では VirtualBox に機能を追加するソフトウェアを製品として開発・販売している．

## 4.2.2 VirtualBox のインストール

Chocolatey のサイトで VirtualBox と入力して検索する。

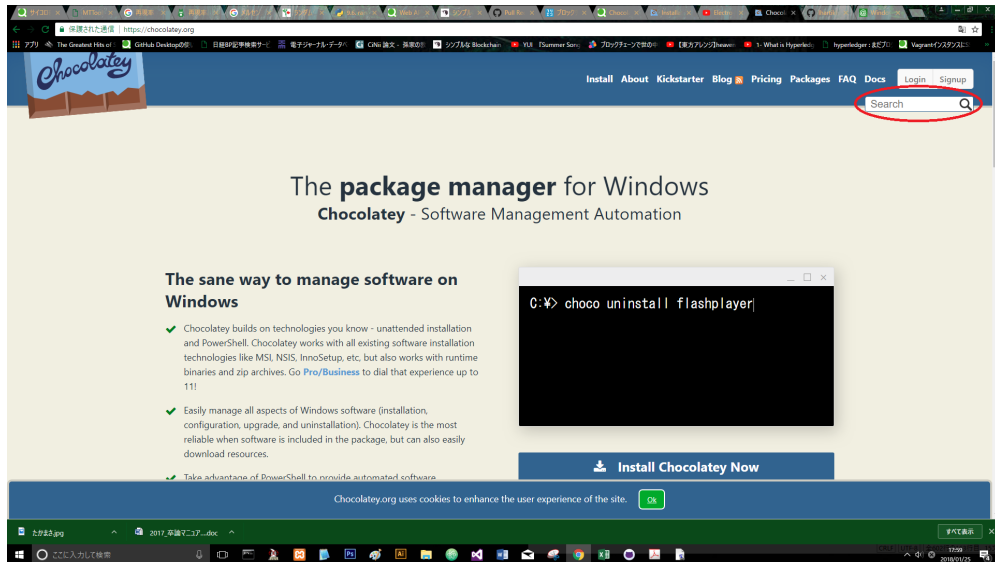


図 4.7 VirtualBox のインストール

検索結果の上位に VirtualBox のパッケージが出てくるので、右側にある黒い部分の virtualbox をインストールするためのコマンドをコピーする。

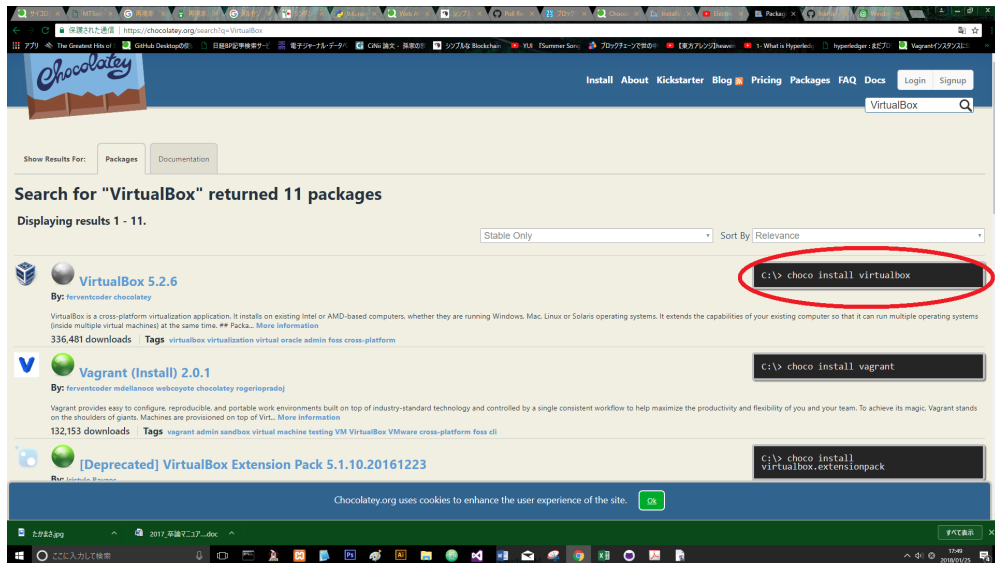


図 4.8 VirtualBox のインストール

コマンドプロンプトを管理者で起動して、コピーした `choco install virtualbox` というコマンドをペーストして実行するとインストールが開始される。

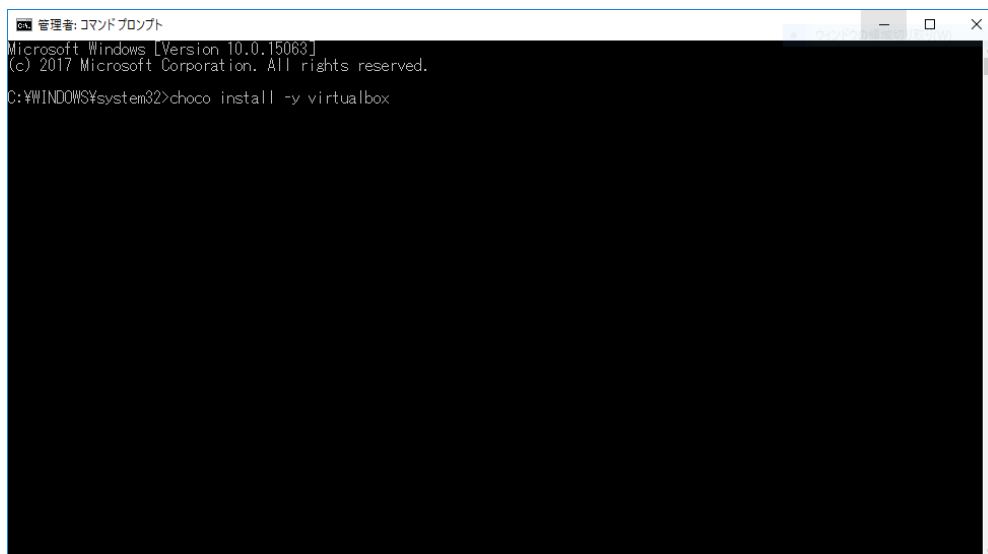


図 4.9 VirtualBox のインストール

また `choco install -y virtualbox` と入力し実行すると一度にインストールが終わる。



```
管理: コマンド プロンプト
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.

C:\Windows\system32>choco install -y virtualbox
Chocolatey v0.10.3
Installing the following packages:
virtualbox
By installing you accept licenses for the packages.
virtualbox v5.0.16.105871 already installed.
Use --force to reinstall, specify a version to install, or try upgrade.

Chocolatey installed 0/1 packages. 0 packages failed.
See the log for details (C:\ProgramData\chocolatey\logs\chocolatey.log).

Warnings:
- virtualbox - virtualbox v5.0.16.105871 already installed.
Use --force to reinstall, specify a version to install, or try upgrade.

C:\Windows\system32>
```

図 4.10 VirtualBox のインストール

このようになればインストール完了。



## 4.3 Vagrant について

### 4.3.1 Vagrant とは

Vagrant とは，仮想環境を作成するにあたって，簡単に構築・管理し配布することができるツールである．

Vagrant を使うには，Box と呼ばれるファイルが必要になる．今回は，矢吹研究室の公式マシンを使用する．

矢吹研究室公式マシンは以下の URL の Github リポジトリで公開されており，このリポジトリを clone して使用する．なお，矢吹研究室公式マシンは Linux ディストリビューションの一つである Ubuntu を使用している．

<https://github.com/yabukilab/machine>

## 4.3.2 Vagrant のインストール

Chocolatey のサイトで Vagrant と入力して検索する。

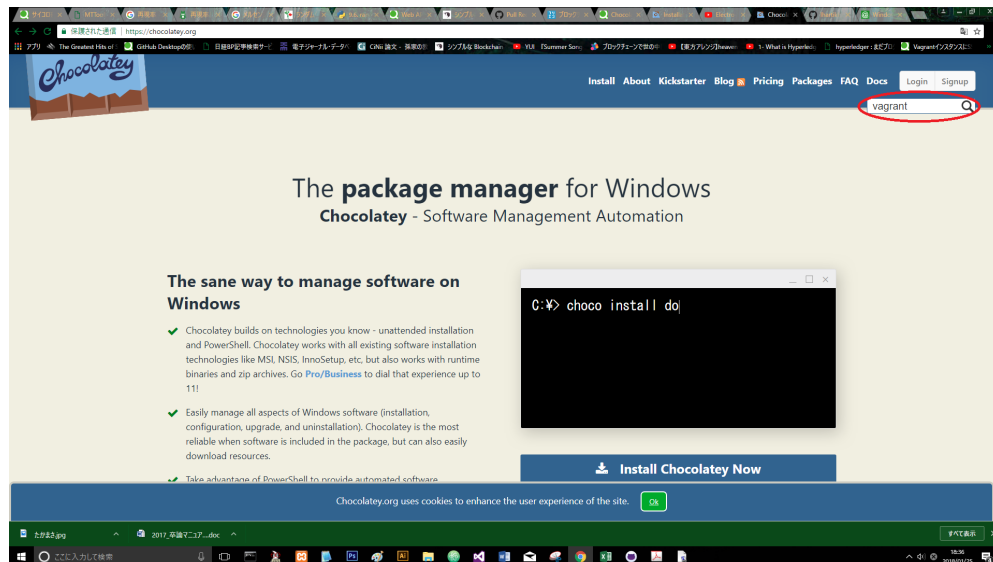


図 4.11 Vagrant のインストール

検索の上位に Vagrant のパッケージが出てくるので、右側にある黒い部分の Vagrant をインストールするためのコマンドをコピーする。

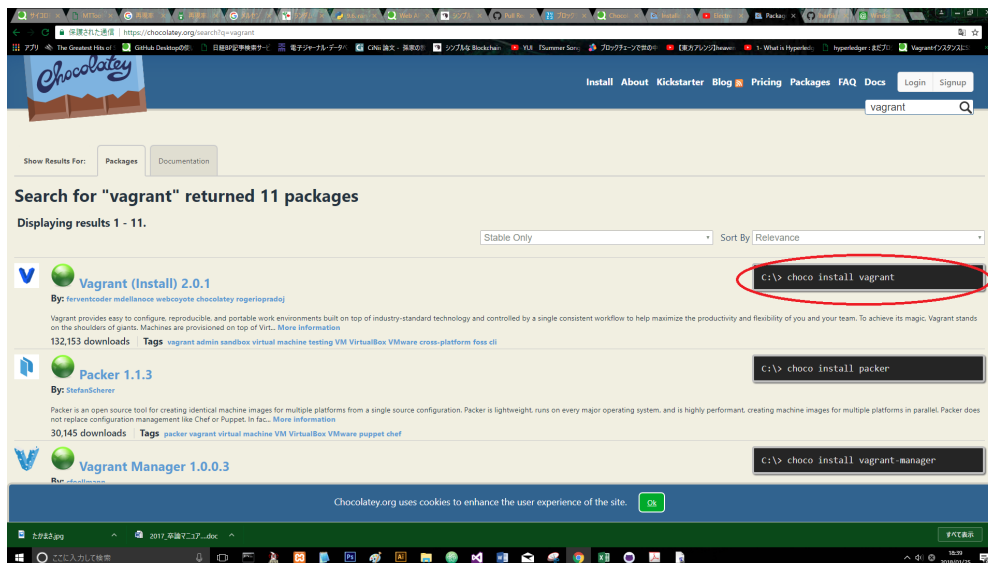


図 4.12 Vagrant のインストール

コマンドプロンプトを管理者で起動して，コピーした `choco install vagrant` というコマンドをペーストして実行するとインストールが開始される。

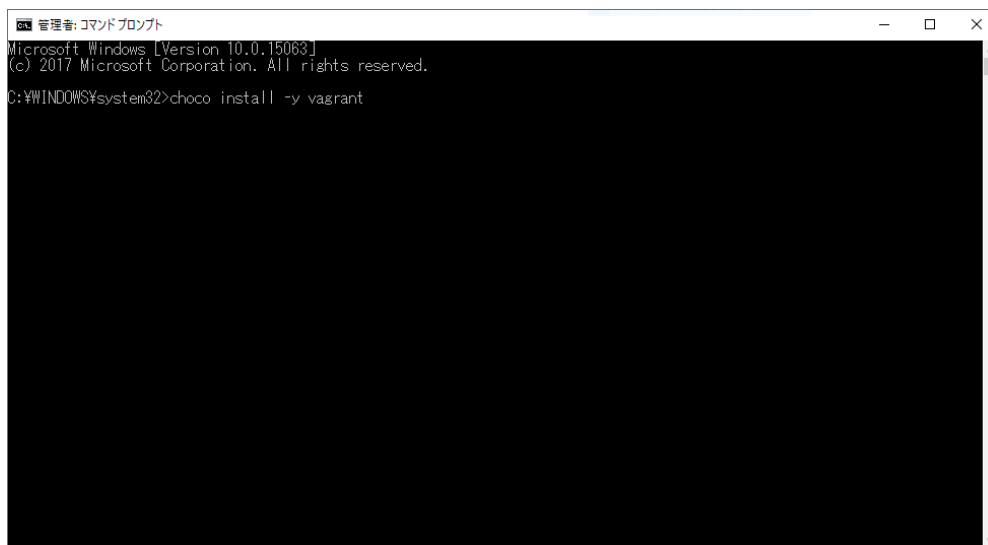
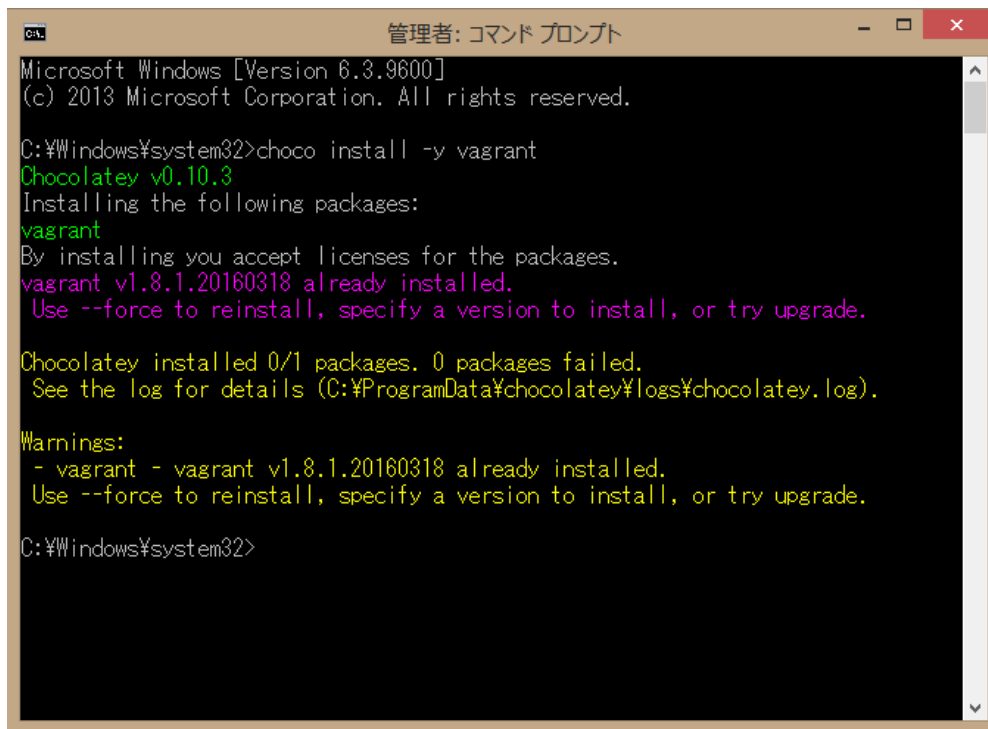


図 4.13 Vagrant のインストール

また、`choco install -y vagrant` と入力し実行すると一度にインストールが終わる。



```
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.

C:\Windows\system32>choco install -y vagrant
Chocolatey v0.10.3
Installing the following packages:
vagrant
By installing you accept licenses for the packages.
vagrant v1.8.1.20160318 already installed.
Use --force to reinstall, specify a version to install, or try upgrade.

Chocolatey installed 0/1 packages. 0 packages failed.
See the log for details (C:\ProgramData\chocolatey\logs\chocolatey.log).

Warnings:
- vagrant - vagrant v1.8.1.20160318 already installed.
Use --force to reinstall, specify a version to install, or try upgrade.

C:\Windows\system32>
```

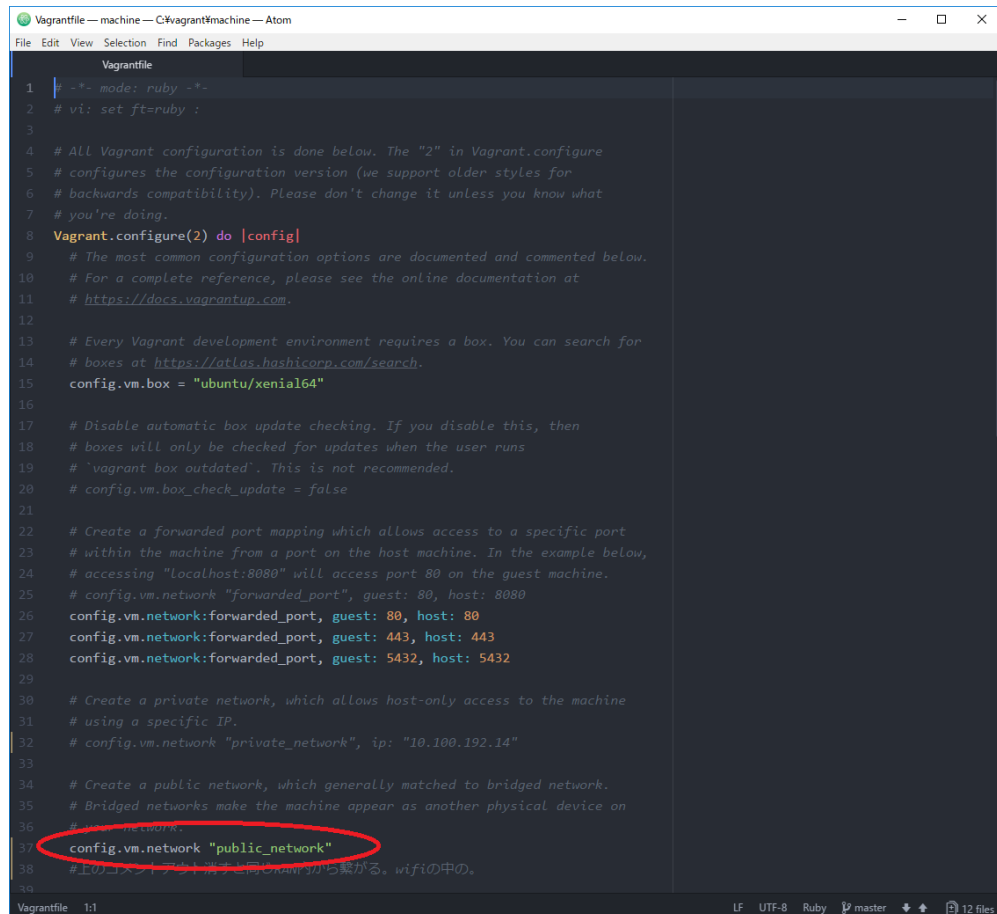
図 4.14 Vagrant のインストール

このようになればインストール成功。

### 4.3.3 データ通信解除

Vagrantfile をテキストエディタで開いて 37 行目にあるコメントアウトを削除する。この行為によって同一ネットワーク上でのデータ通信が解除される。Vagrantfile の場所は以下の通りである。

c:/vagrant/machine/Vagrantfile



```
Vagrantfile
1  -*- mode: ruby -*-
2  # vi: set ft=ruby :
3
4  # All Vagrant configuration is done below. The "2" in Vagrant.configure
5  # configures the configuration version (we support older styles for
6  # backwards compatibility). Please don't change it unless you know what
7  # you're doing.
8  Vagrant.configure(2) do |config|
9    # The most common configuration options are documented and commented below.
10   # For a complete reference, please see the online documentation at
11   # https://docs.vagrantup.com.
12
13   # Every Vagrant development environment requires a box. You can search for
14   # boxes at https://atlas.hashicorp.com/search.
15   config.vm.box = "ubuntu/xenial64"
16
17   # Disable automatic box update checking. If you disable this, then
18   # boxes will only be checked for updates when the user runs
19   # 'vagrant box outdated'. This is not recommended.
20   # config.vm.box_check_update = false
21
22   # Create a forwarded port mapping which allows access to a specific port
23   # within the machine from a port on the host machine. In the example below,
24   # accessing "localhost:8080" will access port 80 on the guest machine.
25   # config.vm.network "forwarded_port", guest: 80, host: 8080
26   config.vm.network:forwarded_port, guest: 80, host: 80
27   config.vm.network:forwarded_port, guest: 443, host: 443
28   config.vm.network:forwarded_port, guest: 5432, host: 5432
29
30   # Create a private network, which allows host-only access to the machine
31   # using a specific IP.
32   # config.vm.network "private_network", ip: "10.100.192.14"
33
34   # Create a public network, which generally matched to bridged network.
35   # Bridged networks make the machine appear as another physical device on
36   # your network.
37   config.vm.network "public_network"
38   #上のコメントアウトを消すと同じネットワークから繋がる。wifiの中の。
39
40 end
```

図 4.15 コメントアウトの削除

#### 4.3.4 計算機環境の構築

Vagrant の Box を用意する．今回は矢吹研究室の公式仮想マシンを用いる．管理者権限のないコマンドプロンプトを起動し，以下のコマンドを入力する．

仮想マシンの再構築を高速にするために，キャッシュを導入する．

```
vagrant plugin install vagrant-cachier
```

仮想マシンを用意する

```
cd \  
mkdir vagrant  
cd vagrant  
git clone https://github.com/yabukilab/machine.git
```

起動

```
cd vagrant
```

```
cd machine
```

```
vagrant up
```

Vagrant が起動に成功したか確認する方法. ブラウザの URL 入力欄に以下の URL を入力してアクセスできるか確認する.

`http://localhost/phpmyadmin/`

ユーザ名 `root`

パスワード `pass`

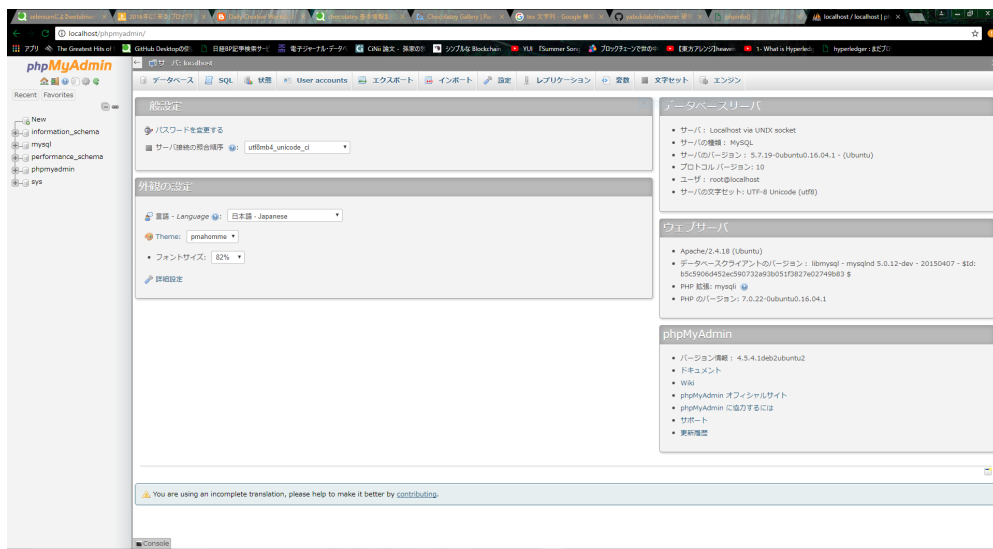


図 4.16 vagrant 起動成功確認画面

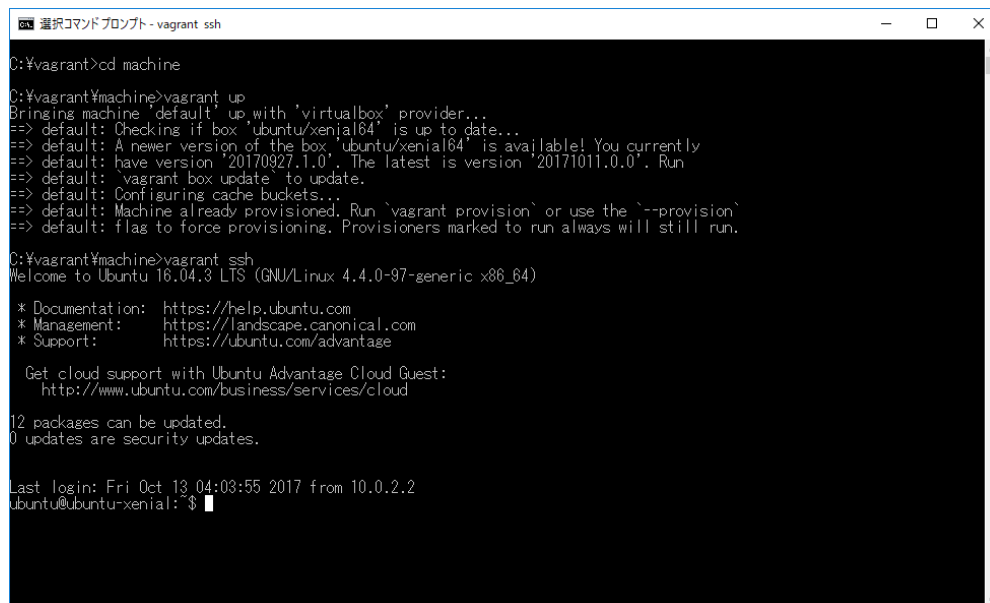
仮想マシンの再起動と停止, 削除.

- ・再起動: `vagrant reload`
- ・停止: `vagrant halt`
- ・削除: `vagrant destroy`



接続

vagrant ssh



```
C:\vagrant>cd machine
C:\vagrant\machine>vagrant up
Bringing machine 'default' up with 'virtualbox' provider...
==> default: Checking if box 'ubuntu/xenial64' is up to date...
==> default: A newer version of the box 'ubuntu/xenial64' is available! You currently
==> default: have version '20170927.1.0'. The latest is version '20171011.0.0'. Run
==> default: 'vagrant box update' to update.
==> default: Configuring cache buckets...
==> default: Machine already provisioned. Run 'vagrant provision' or use the '--provision'
==> default: flag to force provisioning. Provisioners marked to run always will still run.

C:\vagrant\machine>vagrant ssh
Welcome to Ubuntu 16.04.3 LTS (GNU/Linux 4.4.0-97-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

Get cloud support with Ubuntu Advantage Cloud Guest:
http://www.ubuntu.com/business/services/cloud

12 packages can be updated.
0 updates are security updates.

Last login: Fri Oct 13 04:03:55 2017 from 10.0.2.2
ubuntu@ubuntu-xenial: $
```

図 4.17 Vagrantssh 接続

このように表示されれば接続成功。

また、やり直したい場合は以下のコマンドを実行する。

```
c \
cd vagrant
cd machine
cd destroy
cd ..
rmdir /S /Q machine
```

## 4.4 docker について

### 4.4.1 docker とは

docker とはコンテナ型の仮想化環境を提供するオープンソフトウェアである。VMware 製品などの完全仮想化を行うハイパーバイザ型製品と比べて、ディスク使用量が少なく、仮想環境 (インスタンス) 作成や起動は速く、性能劣化がほとんどないという利点を持つ。naivechain を動かすにはまず docker をインストールしなければならない。

### 4.4.2 リソースの効率化

一台のサーバ上に複数のオペレーティングシステム (OS) を走らせる仮想化技術は従来から存在していた。例えばハイパーバイザ型の Hyper-V やホスト型の VirtualBox 等である。仮想化の本来の目的は、一台のハードウェア内に出来る限り多くのアプリケーションを実行する事であるが、上記種類の仮想化では仮想環境毎に OS を丸ごとインストールする必要がある、アプリケーションに必要なサービスやファイルまで伴っていた。これはリソースの無駄使いであった。

アプリケーションとは直接関係の無いライブラリやデータは仮想環境内で共有する事が望ましかった。これを実現するのがコンテナ型の仮想化である。実際に Docker は、ホスト OS のカーネルを共有する。それぞれの仮想環境は Docker コンテナと呼ばれ、一台のサーバ上でそれぞれが隔離される事で複数のインスタンスが動作しているように見える。

### 4.4.3 アプリ実行環境の容易さ

一般にアプリケーションを開発もしくは動作させるまでには、設定ファイルの編集や必要なライブラリのインストール等、本来の目的には関係のない煩雑な作業が必要である。Docker はアプリケーションとライブラリを同一のコンテナ内に固めてしまう。一度固めたコンテナは軽量であるため移動が容易であり、比較的どの環境でも素早く目的のアプリケーションを動作させる事が可能である。これを Docker 社は、Build, Ship, and Run Any App, Anywhere と表現している [3]。

### 4.4.4 docker の評価

Docker のコンテナ管理の手軽さやインスタンス操作の高速性は、クラウドサービスやビッグデータ基盤などを管理するための IT 基盤として高く評価され、2014 年 12 月日経 BP 社より「IT インフラテクノロジー AWARD 2015」グランプリに選出されている。

日本では「Immutable Infrastructure (不変のインフラ)」という表現と共に話題になることが多い。

2014 年、Google は Docker とは言及していないが、コンテナ型仮想技術を利用しており、毎週 20 億個のコンテナを自社サービスのために起動していると発表した。

#### 4.4.5 docker のインストール

上記の `vagrant ssh` を行って仮想マシンに接続した状態で `docker` のインストールを行う。  
以下のコマンドを実行する。

```
sudo apt-get install docker.io -y
```

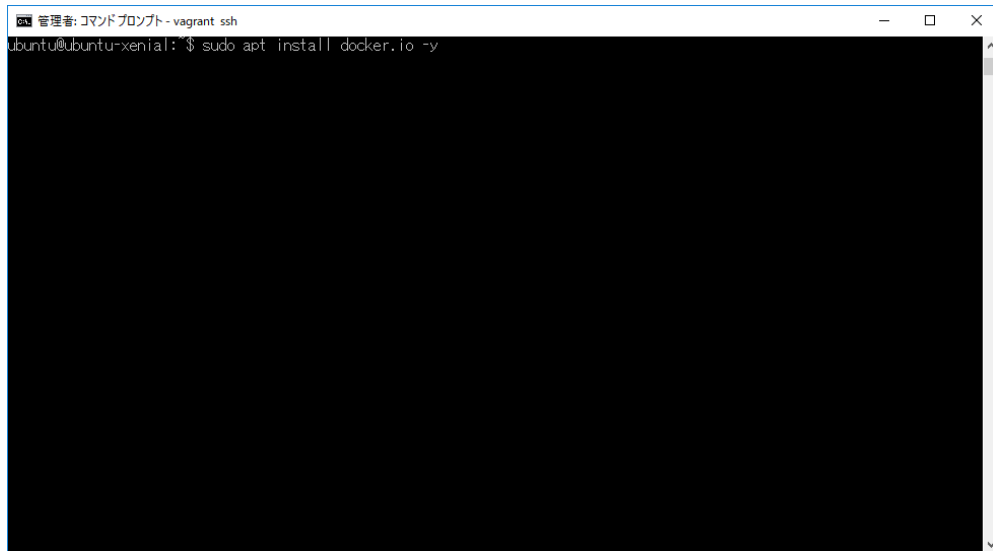


図 4.18 docker のインストール

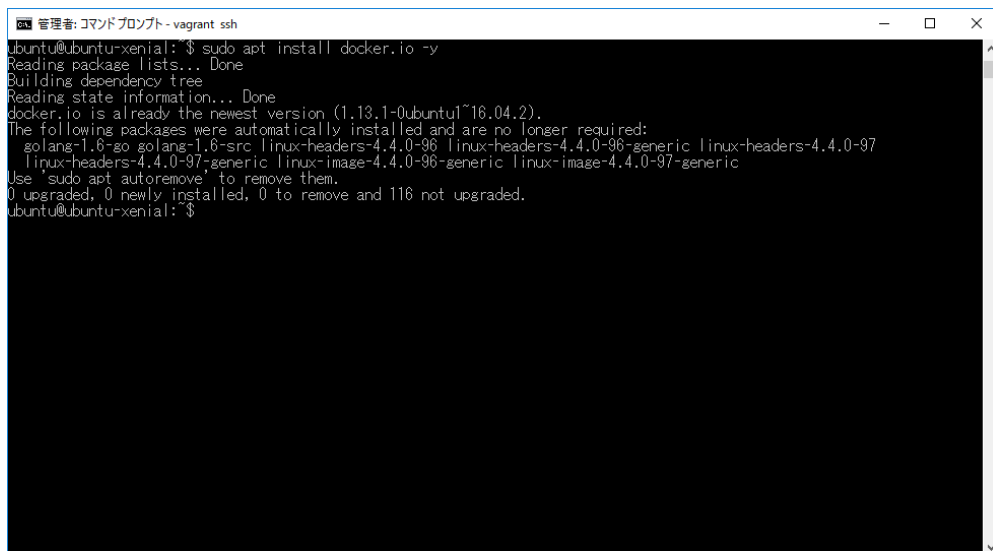
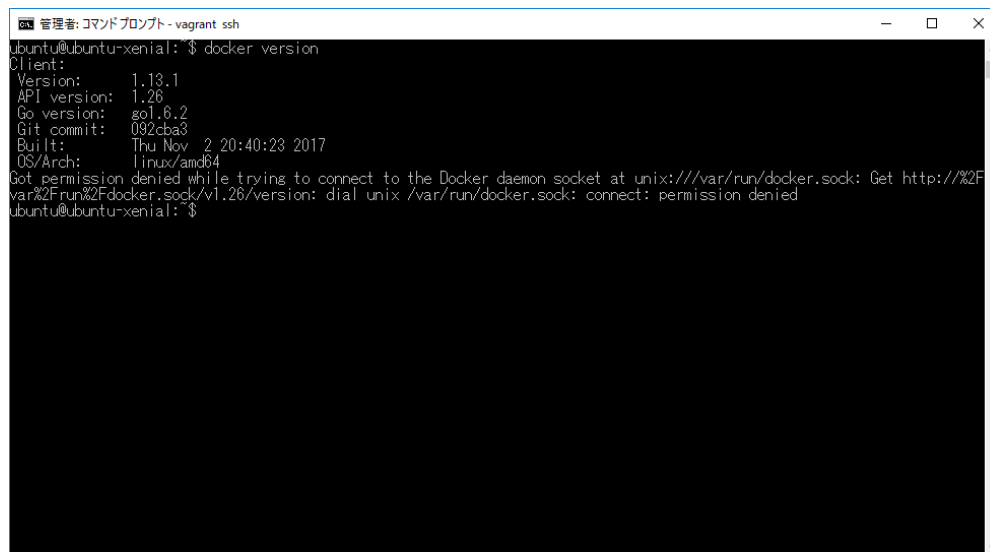


図 4.19 docker のインストール

このように表示されればインストール成功。

docker のバージョンを確認する場合は以下のコマンドを実行する。

`docker version`



```
管理: コマンドプロンプト - vagrant ssh
ubuntu@ubuntu-xenial:~$ docker version
Client:
Version:      1.13.1
API version:  1.26
Go version:   go1.6.2
Git commit:   092c3a3
Built:        Thu Nov  2 20:40:23 2017
OS/Arch:      linux/amd64
Got permission denied while trying to connect to the Docker daemon socket at unix:///var/run/docker.sock: Get http://%2Fvar%2Frun%2Fdocker.sock/v1.26/version: dial unix /var/run/docker.sock: connect: permission denied
ubuntu@ubuntu-xenial:~$
```

図 4.20 docker のバージョン確認

このように表示されれば docker の version がわかる。

## 4.5 docker compose について

### 4.5.1 docker compose とは

docker compose とは docker コンテナの管理を支援する純正ツールである。Docker Compose は Docker が開発するコマンドラインツールで、あらかじめ用意しておいた設定ファイルに従ってコンテナを起動するツールだ。設定ファイルには複数のコンテナに関する記述が可能で、コンテナの起動オプションやコンテナに与える環境変数など、さまざまな設定も同時に記述できる [4]。

また、コンテナ同士の依存関係を設定することも可能で、これによって関連するコンテナを複数まとめて起動することも可能だ。

ここでこの画像を入れる

### 4.5.2 docker compose の利用ケース

Docker Compose を活用できるケースとしては、まずソフトウェアの開発/テスト環境が挙げられる。昨今ではデータベースやキャッシュ、フロントエンドといった役割毎に複数のサーバーを用意し、それらを組み合わせてサービスを構築する例が多い。こういった場合、開発/テスト環境でもそれらをすべて用意する必要がある。それらをコンテナの形で用意しておき、それらを起動するためのオプション設定などを Docker Compose の設定ファイルに記述しておけば、コマンドを 1 つ実行するだけでテスト環境を立ち上げられるようになる。

また、7 月末にリリースされた Docker 1.12 では「docker stack」という機能が実験的に導入された。これを利用することで、Docker Compose 向けに定義したコンテナの設定等をそのまま利用して Docker クラスタ内でコンテナを稼働させられるようになる。これにより、Docker Compose を使って作ったテスト環境を、容易に実運用環境に移行させられるようになることが期待される。

### 4.5.3 Docker Compose のインストール方法

docker のインストールが終わっている仮想マシンで以下のコマンドを実行する。

```
sudo apt-get install docker-compose -y
```

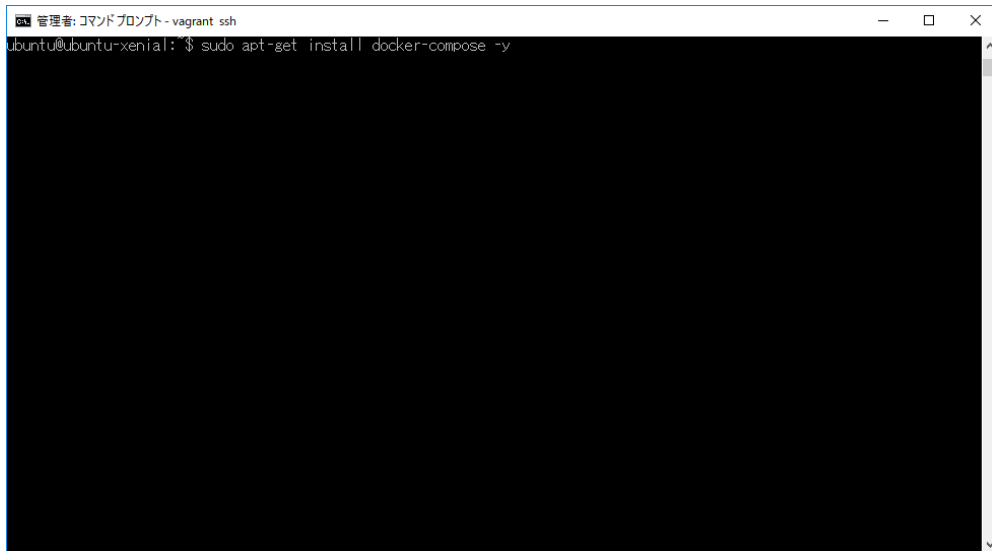


図 4.21 docker compose のインストール

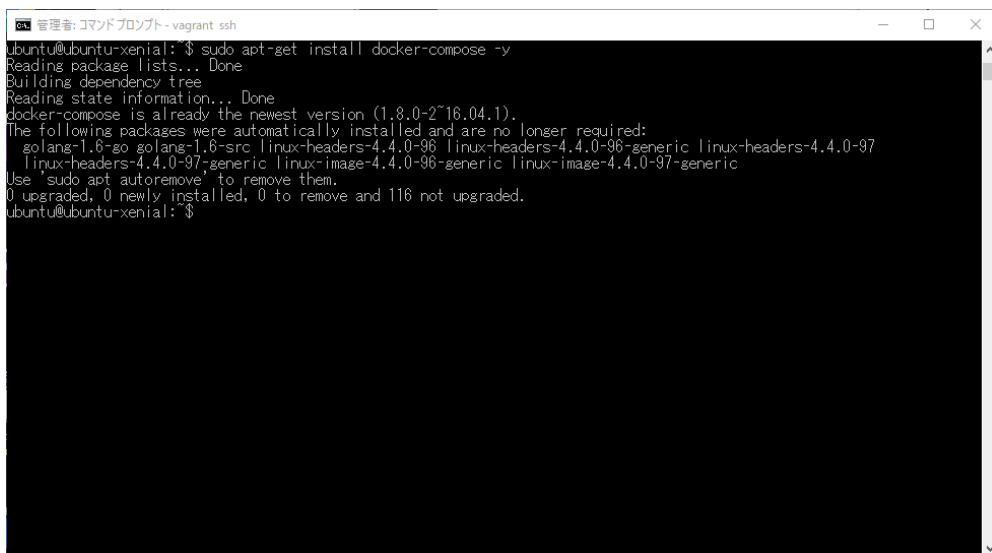


図 4.22 docker compose のインストール

このように表示されれば docker compose のインストール成功.

docker compose のバージョンを確認する場合は以下のコマンドを実行する.

`docker-compose -v`

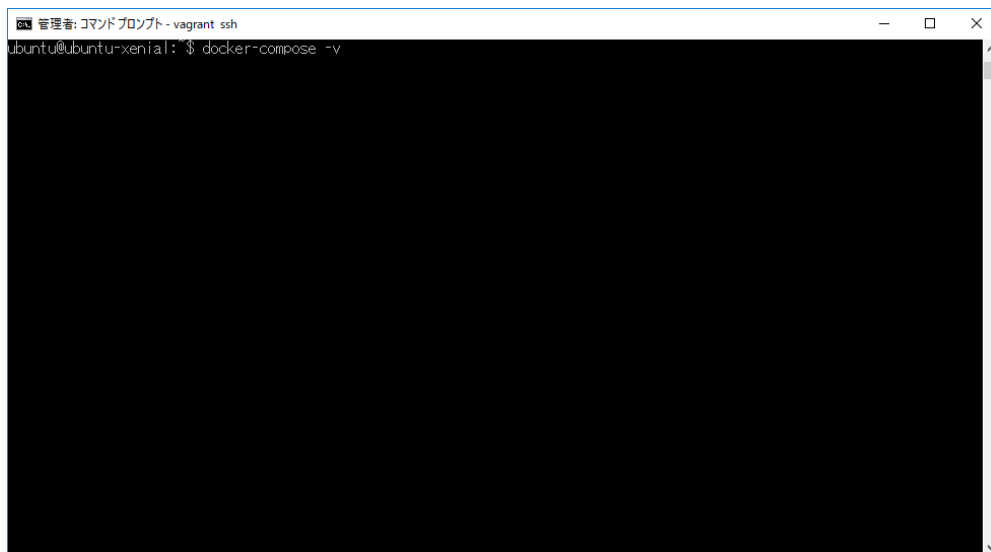


図 4.23 docker compose のバージョン確認

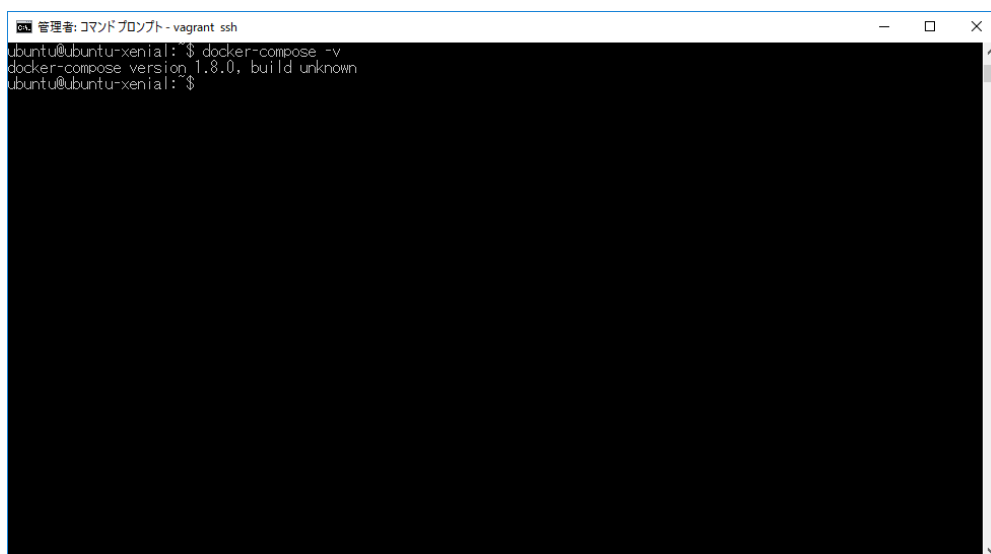


図 4.24 docker compose のバージョン確認

このように表示されれば docker-compose の version がわかる.

## 4.6 naivechain について

### 4.6.1 naivechain とは

ブロックと呼ばれるデータが順序つけられた 200 行の javascript に実装した非常にシンプルなブロックチェーンである。naivechain ではインデックス、タイムスタンプ、データ、ハッシュ値、そして一つ前のブロックのハッシュ値のみの構造である。



図 4.25 naivechain のイメージ



## 4.6.2 パッケージリストの更新

まず、先に述べた仮想マシンに接続した状態から以下のコマンドでパッケージリストの更新を行う。 `sudo apt-get update`

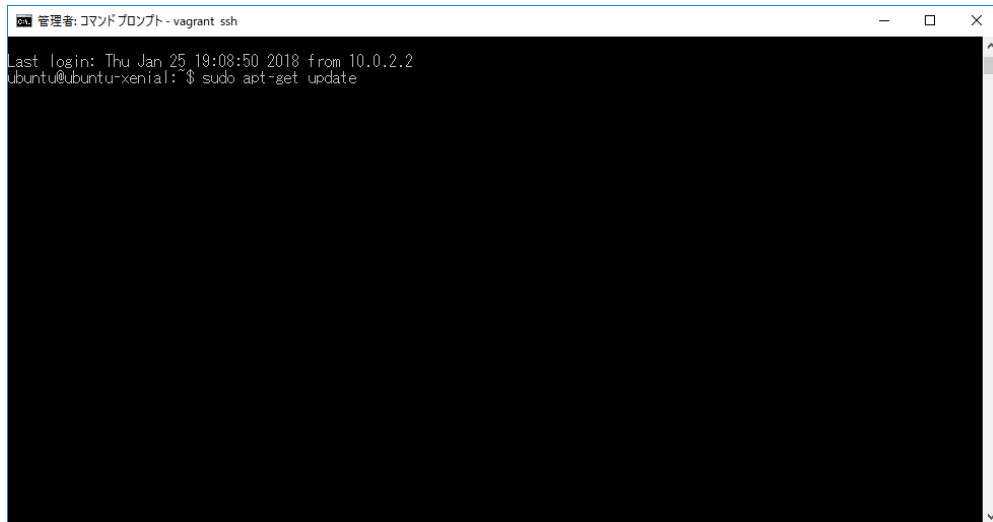


図 4.26 パッケージリストの更新

パッケージリストのアップデート完了画面。

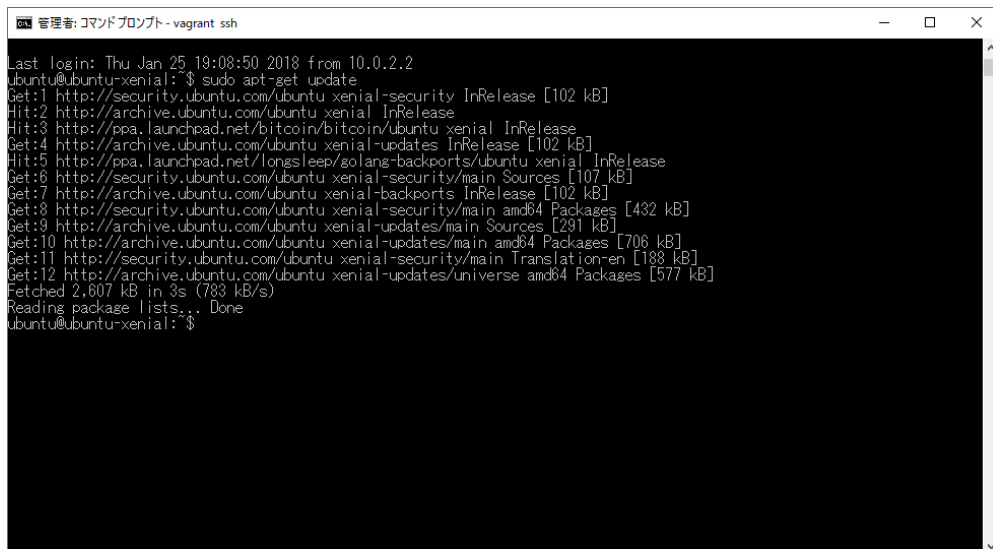


図 4.27 パッケージリストの更新

### 4.6.3 naivechain のインストール

docker, docker compose のインストールが終わっている仮想マシンで以下のコマンドを実行する.

```
git clone https://github.com/lhartikk/naivechain
```

naivechain のファイルへ移動するために以下のコマンドを実行する.

```
cd naivechain
```

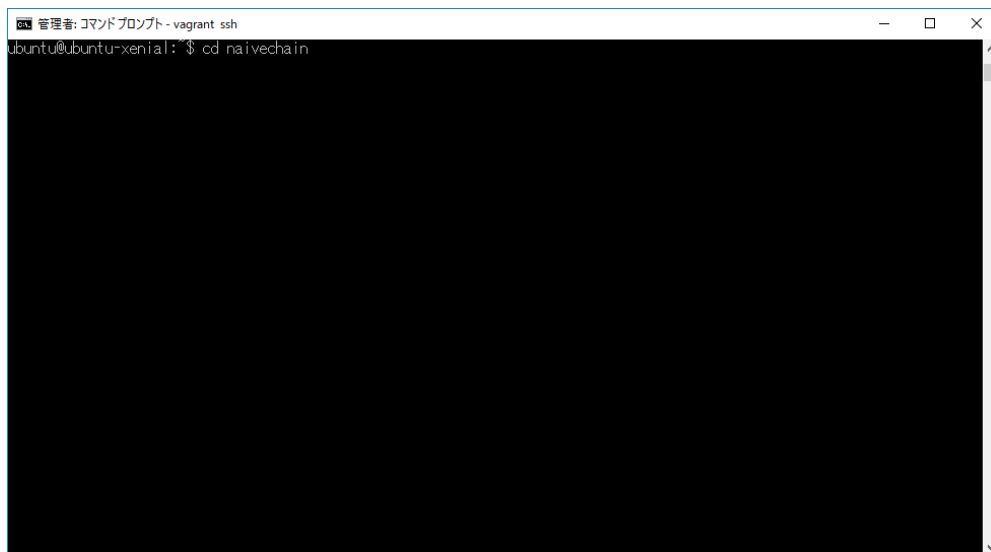


図 4.28 パッケージリストの更新

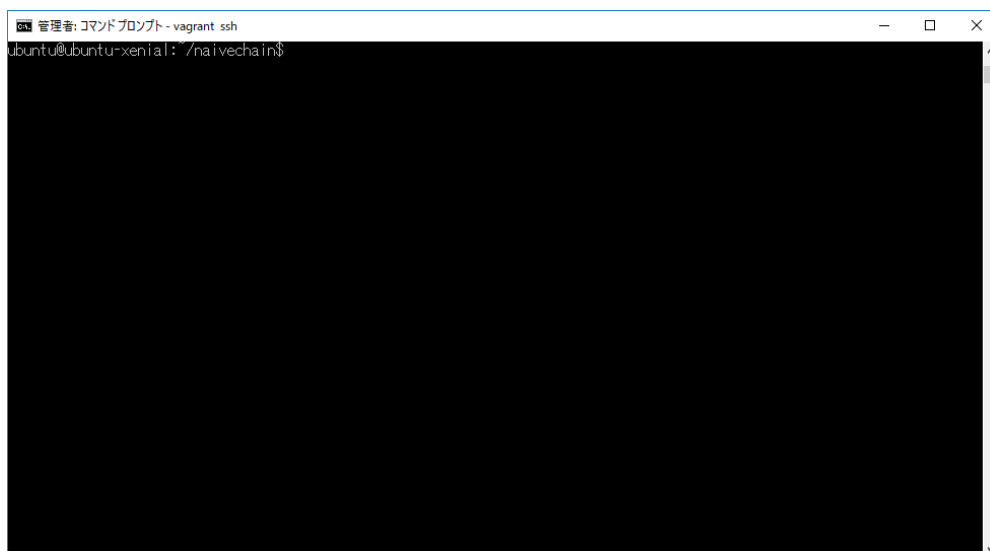


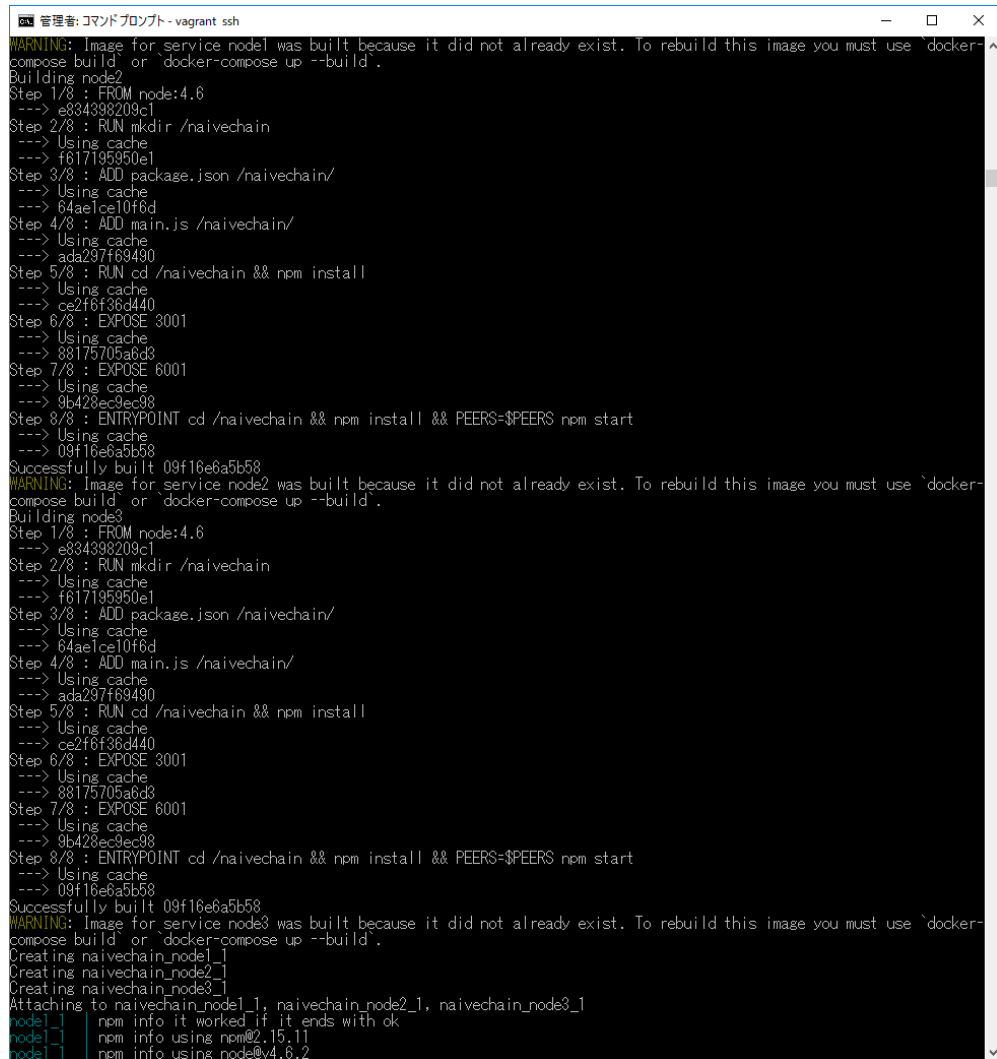
図 4.29 パッケージリストの更新

このように表示されれば **naivechain** ファイルへの移動成功である。

## 4.6.4 naivechain の起動

naivechain を実行するために以下のコマンドを実行する。

```
sudo docker-compose up
```



```
WARNING: Image for service node1 was built because it did not already exist. To rebuild this image you must use `docker-
compose build` or `docker-compose up --build`.
Building node2
Step 1/8 : FROM node:4.6
--> e834398209c1
Step 2/8 : RUN mkdir /naivechain
--> Using cache
--> f617195950e1
Step 3/8 : ADD package.json /naivechain/
--> Using cache
--> 64ae1ce10f6d
Step 4/8 : ADD main.js /naivechain/
--> Using cache
--> ada297f69490
Step 5/8 : RUN cd /naivechain && npm install
--> Using cache
--> ce2f6f36d440
Step 6/8 : EXPOSE 3001
--> Using cache
--> 88175705a6d3
Step 7/8 : EXPOSE 6001
--> Using cache
--> 9b428ec9ec98
Step 8/8 : ENTRYPOINT cd /naivechain && npm install && PEERS=$PEERS npm start
--> Using cache
--> 09f16e6a5b58
Successfully built 09f16e6a5b58
WARNING: Image for service node2 was built because it did not already exist. To rebuild this image you must use `docker-
compose build` or `docker-compose up --build`.
Building node3
Step 1/8 : FROM node:4.6
--> e834398209c1
Step 2/8 : RUN mkdir /naivechain
--> Using cache
--> f617195950e1
Step 3/8 : ADD package.json /naivechain/
--> Using cache
--> 64ae1ce10f6d
Step 4/8 : ADD main.js /naivechain/
--> Using cache
--> ada297f69490
Step 5/8 : RUN cd /naivechain && npm install
--> Using cache
--> ce2f6f36d440
Step 6/8 : EXPOSE 3001
--> Using cache
--> 88175705a6d3
Step 7/8 : EXPOSE 6001
--> Using cache
--> 9b428ec9ec98
Step 8/8 : ENTRYPOINT cd /naivechain && npm install && PEERS=$PEERS npm start
--> Using cache
--> 09f16e6a5b58
Successfully built 09f16e6a5b58
WARNING: Image for service node3 was built because it did not already exist. To rebuild this image you must use `docker-
compose build` or `docker-compose up --build`.
Creating naivechain_node1_1
Creating naivechain_node2_1
Creating naivechain_node3_1
Attaching to naivechain_node1_1, naivechain_node2_1, naivechain_node3_1
node1_1 | npm info it worked if it ends with ok
node1_1 | npm info using npm@2.15.11
node1_1 | npm info using node@v4.6.2
```

図 4.30 compose up 成功画面

```
管理: コマンドプロンプト - vagrant ssh
node1 | npm WARN package.json naivechain@1.0.0 No description
node1 | npm WARN package.json naivechain@1.0.0 No repository field.
node1 | npm WARN package.json naivechain@1.0.0 No README data
node1 | npm WARN package.json naivechain@1.0.0 No license field.
node1 | npm info preinstall naivechain@1.0.0
node1 | npm info build /naivechain
node1 | npm info linkStuff naivechain@1.0.0
node1 | npm info install naivechain@1.0.0
node1 | npm info postinstall naivechain@1.0.0
node1 | npm info prepublish naivechain@1.0.0
node2 | npm info it worked if it ends with ok
node2 | npm info using npm@2.15.11
node2 | npm info using node@v4.6.2
node3 | npm info it worked if it ends with ok
node3 | npm info using npm@2.15.11
node3 | npm info using node@v4.6.2
node1 | npm info ok
node2 | npm WARN package.json naivechain@1.0.0 No description
node2 | npm WARN package.json naivechain@1.0.0 No repository field.
node2 | npm WARN package.json naivechain@1.0.0 No README data
node2 | npm WARN package.json naivechain@1.0.0 No license field.
node2 | npm info preinstall naivechain@1.0.0
node3 | npm WARN package.json naivechain@1.0.0 No description
node3 | npm WARN package.json naivechain@1.0.0 No repository field.
node3 | npm WARN package.json naivechain@1.0.0 No README data
node3 | npm WARN package.json naivechain@1.0.0 No license field.
node3 | npm info preinstall naivechain@1.0.0
node3 | npm info build /naivechain
node3 | npm info linkStuff naivechain@1.0.0
node1 | npm info it worked if it ends with ok
node1 | npm info using npm@2.15.11
node1 | npm info using node@v4.6.2
node3 | npm info install naivechain@1.0.0
node3 | npm info postinstall naivechain@1.0.0
node2 | npm info build /naivechain
node3 | npm info prepublish naivechain@1.0.0
node2 | npm info linkStuff naivechain@1.0.0
node2 | npm info install naivechain@1.0.0
node2 | npm info postinstall naivechain@1.0.0
node2 | npm info prepublish naivechain@1.0.0
node2 | npm info ok
node3 | npm info ok
node1 | npm info prestart naivechain@1.0.0
node1 | npm info start naivechain@1.0.0
node1 | > naivechain@1.0.0 start /naivechain
node1 | > node main.js
node2 | npm info it worked if it ends with ok
node2 | npm info using npm@2.15.11
node2 | npm info using node@v4.6.2
node3 | npm info it worked if it ends with ok
node3 | npm info using npm@2.15.11
node3 | npm info using node@v4.6.2
node2 | npm info prestart naivechain@1.0.0
node2 | npm info start naivechain@1.0.0
node2 | > naivechain@1.0.0 start /naivechain
node2 | > node main.js
node1 | listening websocket p2p port on: 6001
node1 | Listening http on port: 3001
node1 | npm info prestart naivechain@1.0.0
```

図 4.31 compose up 成功画面

```
naivechain@1.0.0
> naivechain@1.0.0 start /naivechain
> node main.js

listening websocket p2p port on: 6001
Listening http on port: 3001
Received message["type":0]
Received message["type":0]
Received message["type":2,"data":{"index":0,"previousHash":"","timestamp":1465154705,"data":"my
genesis block!!","hash":"816534932c2b7154836da6afc367695e6337db8a921823784c14378abed4f7d7"}]]
received blockchain is not longer than received blockchain. Do nothing
Received message["type":2,"data":{"index":0,"previousHash":"","timestamp":1465154705,"data":"my
genesis block!!","hash":"816534932c2b7154836da6afc367695e6337db8a921823784c14378abed4f7d7"}]]
received blockchain is not longer than received blockchain. Do nothing
listening websocket p2p port on: 6001
Listening http on port: 3001
Received message["type":0]
Received message["type":0]
Received message["type":2,"data":{"index":0,"previousHash":"","timestamp":1465154705,"data":"my
genesis block!!","hash":"816534932c2b7154836da6afc367695e6337db8a921823784c14378abed4f7d7"}]]
received blockchain is not longer than received blockchain. Do nothing
Received message["type":2,"data":{"index":0,"previousHash":"","timestamp":1465154705,"data":"my
genesis block!!","hash":"816534932c2b7154836da6afc367695e6337db8a921823784c14378abed4f7d7"}]]
received blockchain is not longer than received blockchain. Do nothing
```

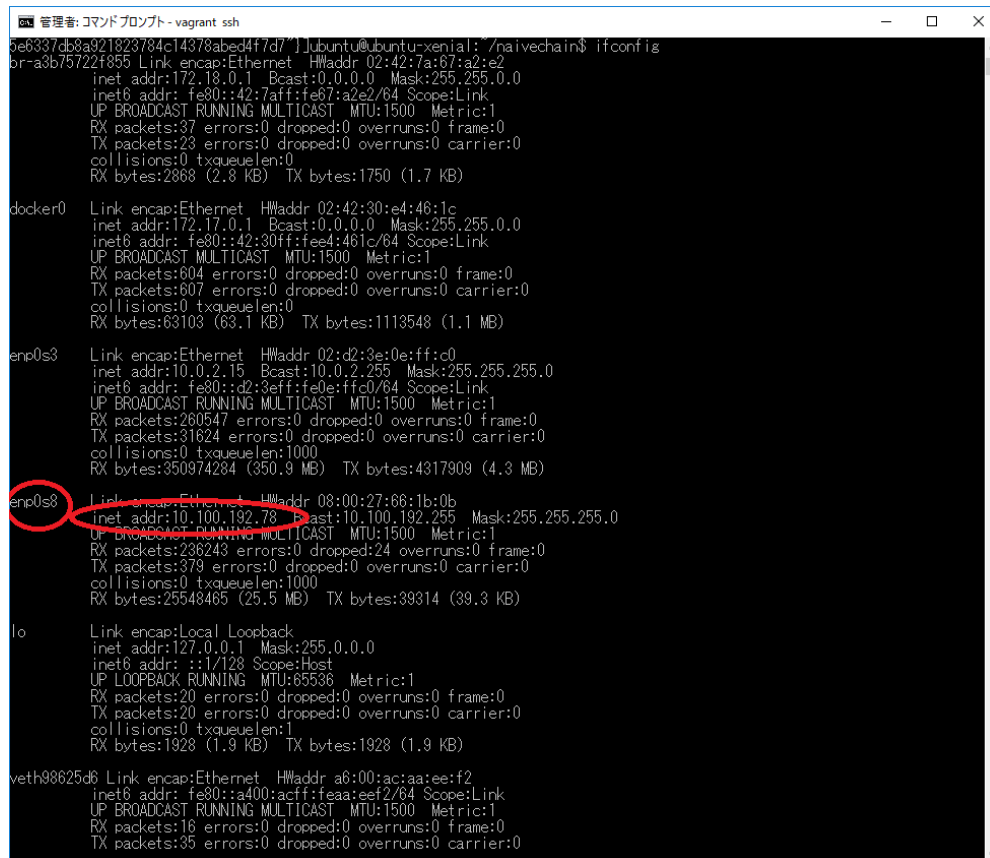
図 4.32 compose up 成功画面

このように表示されれば docker compose の起動成功である。naivechain を使用することができる。

## 4.6.5 ip アドレス確認

naivechain を用いる際に、自分のノードがホストノードとなって他ノードから自分のノードへブロックを追加する場合、自分の ip アドレスを確認する必要がある。そのため以下のコマンドを実行して ip アドレスを確認する必要がある。

ifconfig



```
be6337db8a921823784c14378abed4f7d/] Ubuntu@ubuntu-xenial: /naivechain$ ifconfig
br-a3b75722f855 Link encap:Ethernet HWaddr 02:42:7a:67:a2:e2
  inet addr:172.18.0.1 Bcast:0.0.0.0 Mask:255.255.0.0
  inet6 addr: fe80::42:7aff:fe67:a2e2/64 Scope:Link
  UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
  RX packets:37 errors:0 dropped:0 overruns:0 frame:0
  TX packets:23 errors:0 dropped:0 overruns:0 carrier:0
  collisions:0 txqueuelen:0
  RX bytes:2868 (2.8 KB) TX bytes:1750 (1.7 KB)

docker0 Link encap:Ethernet HWaddr 02:42:30:e4:46:1c
  inet addr:172.17.0.1 Bcast:0.0.0.0 Mask:255.255.0.0
  inet6 addr: fe80::42:30ff:fe4:461c/64 Scope:Link
  UP BROADCAST MULTICAST MTU:1500 Metric:1
  RX packets:604 errors:0 dropped:0 overruns:0 frame:0
  TX packets:607 errors:0 dropped:0 overruns:0 carrier:0
  collisions:0 txqueuelen:0
  RX bytes:63103 (63.1 KB) TX bytes:1113548 (1.1 MB)

enp0s3 Link encap:Ethernet HWaddr 02:d2:3e:0e:ff:c0
  inet addr:10.0.2.15 Bcast:10.0.2.255 Mask:255.255.255.0
  inet6 addr: fe80::d2:3eff:fe0e:ffc0/64 Scope:Link
  UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
  RX packets:260547 errors:0 dropped:0 overruns:0 frame:0
  TX packets:31624 errors:0 dropped:0 overruns:0 carrier:0
  collisions:0 txqueuelen:1000
  RX bytes:350974284 (350.9 MB) TX bytes:4317909 (4.3 MB)

enp0s8 Link encap:Ethernet HWaddr 08:00:27:66:1b:0b
  inet addr:10.100.192.78 Bcast:10.100.192.255 Mask:255.255.255.0
  UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
  RX packets:236249 errors:0 dropped:24 overruns:0 frame:0
  TX packets:379 errors:0 dropped:0 overruns:0 carrier:0
  collisions:0 txqueuelen:1000
  RX bytes:25548465 (25.5 MB) TX bytes:39314 (39.3 KB)

lo Link encap:Local Loopback
  inet addr:127.0.0.1 Mask:255.0.0.0
  inet6 addr: ::1/128 Scope:Host
  UP LOOPBACK RUNNING MTU:65536 Metric:1
  RX packets:20 errors:0 dropped:0 overruns:0 frame:0
  TX packets:20 errors:0 dropped:0 overruns:0 carrier:0
  collisions:0 txqueuelen:1
  RX bytes:1928 (1.9 KB) TX bytes:1928 (1.9 KB)

veth98625d6 Link encap:Ethernet HWaddr a6:00:ac:aa:ee:f2
  inet6 addr: fe80::a400:acff:feaa:ee2/64 Scope:Link
  UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
  RX packets:16 errors:0 dropped:0 overruns:0 frame:0
  TX packets:35 errors:0 dropped:0 overruns:0 carrier:0
```

図 4.33 ip アドレス確認

enp0s8

inet addr:10.100.192.78

赤い丸で囲った部分が自分の ip アドレスである。

## 4.7 Bitcoin Core について

Bitcoin Core は Bitcoin クライアントのリファレンス実装であり，公式リポジトリから誰でもダウンロードして利用することができる．ビットコインの BTC は取引所を介してインターネット上で売買されているため，本プログラムを使用すれば，本物のビットコインの送金や，マイニングが可能になる [5]．

ここでは Oracle 社製の仮想ソフトウェアである VirtualBox を使用して Windows をホスト OS とした環境上にゲスト OS として Ubuntu を導入し，その上にブロックチェーン環境を構築する．

### 4.7.1 Bitcoin Core のソースコード・ライブラリの取得

```
mkdir src
cd src
git clone https://github.com/bitcoin/bitcoin.git
```

### 4.7.2 パッケージリストの更新

```
sudo apt-get update
```



### 4.7.3 gcc をインストール

```
sudo apt-get install build-essential libtool autotools-dev automake pkg-config  
libssl-dev libevent-dev bsdmainutils
```

### 4.7.4 OpenSSL をインストール

```
sudo apt-get install libtool autotools-dev autoconf  
sudo apt-get install libssl-dev
```

### 4.7.5 gcc をインストール

```
sudo apt-get install libboost-all-dev
```

### 4.7.6 libd4.8 をインストール

```
sudo apt-get-repository ppa:bitcoin/bitcoin  
sudo apt-get update  
sudo apt-get install libd4.8-dev  
sudo apt-get install libqrencode-dev
```

### 4.7.7 関連ライブラリをインストール

```
sudo apt-get install libminiupnpc-dev  
sudo apt-get install libqrencode-dev
```

### 4.7.8 GUI ライブラリをインストール

```
sudo apt-get install libqt5gui5 libqt5core5a libqt5dbus5  
qttools5-dev qttools5-dev-tools  
libprotobuf-dev protobuf-compiler
```

#### 4.7.9 Bitcoin Core のビルド

```
cd bitcoin  
./autogen.sh  
./configure  
make
```

#### 4.7.10 関連ライブラリをインストール

```
sudo make install
```

#### 4.7.11 bitcoind の起動

```
bitcoind -regtest -daemon
```

#### 4.7.12 bitcoin-cli の起動

```
bitcoin-cli -regtest stop
```

## 第 5 章

# 結果

naivechain と Bitcoin Core の 2 つのブロックチェーンシステムを用いて研究を進めた。サイコロを振った結果をブロックに書き込み，複数の端末機器の間で結果を共有できるアプリケーションを作成する事を目標に研究を進める。naivechain のプログラムを入れているノードからでも，同一ネットワーク上で繋がっているノードであればデータを共有する事が可能になった。ホストノードの URL を指定することで，ブロックチェーン上の全ブロックの閲覧と，ホストノード上にあるブロックチェーンにブロックを作成して追加する事ができた。

Bitcoin Core では複数のアカウントを作成し，アカウント間でのビットコインの送金が可能になった。

## 5.1 naivechain による P2P ネットワーク

### 5.1.1 自分のノードからのブロックの作成とブロックチェーンへの追加

以下のコマンドを実行すれば仮想マシンと、naivechain の入っている自分のノードからブロックチェーン上にブロックを追加することができる。

```
curl -H "Content-type:application/json" --data
'{"data" : "Some data to the first block"}'
http://localhost:3001/mineBlock

'{"data" : "Some data to the first block"}'
```

上記のコマンドのうちの""で閉じられている, "block"と"Some data to the first block"は自由に変更してブロックに追加することができる。今回は"Some data to the first block"の部分を"一番目のブロックです"と記述してブロックチェーンに追加した。

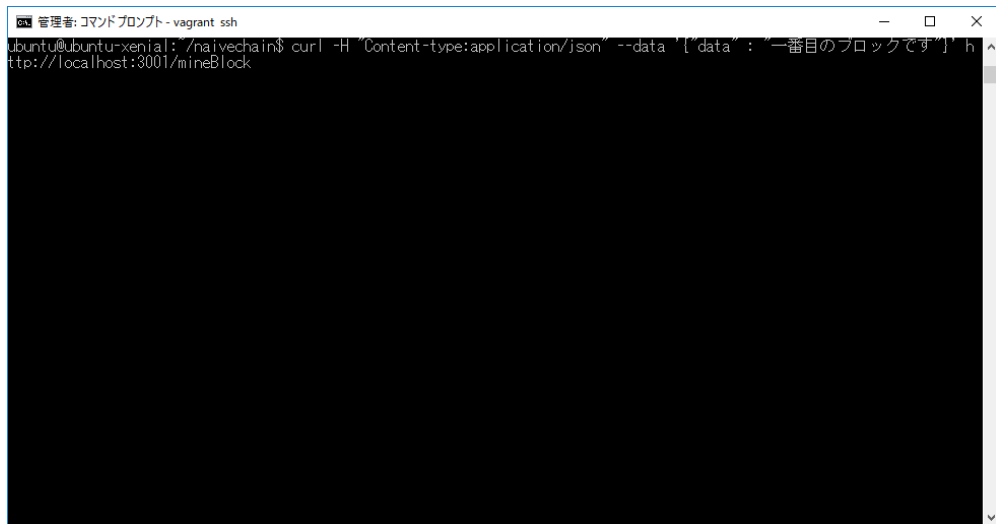
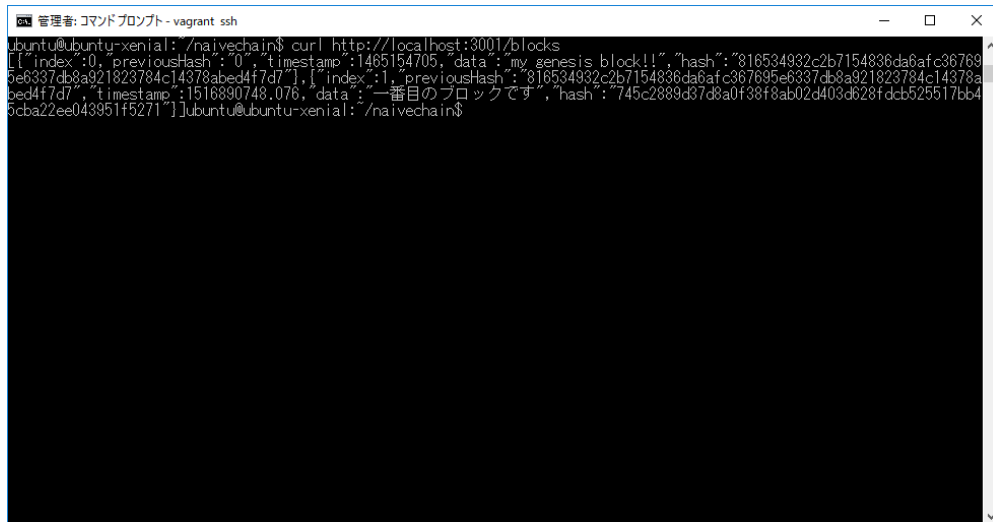


図 5.1 ブロックチェーンへのブロック追加

### 5.1.2 ブロックの確認

下記のコマンドを実行した場合、全ブロックチェーンを一覧することができる。

```
curl http://localhost:3001/blocks
```

A terminal window titled "管理者: コマンドプロンプト - vagrant ssh" showing the output of the command `curl http://localhost:3001/blocks`. The output is a JSON array of two blocks. The first block is the genesis block with index 0, previous hash 0, timestamp 1465154705, and data "my genesis block!!". The second block has index 1, previous hash of the first block, timestamp 1516890748.076, and data "一番目のブロックです".

```
ubuntu@ubuntu-xenial: /naivechain$ curl http://localhost:3001/blocks
[[{"index":0,"previousHash":"0","timestamp":1465154705,"data":"my genesis block!!","hash":"816534932c2b7154836da6afc36769
6e6337db8a921823784c14378abed4f7d7"},{"index":1,"previousHash":"816534932c2b7154836da6afc367696e6337db8a921823784c14378a
bed4f7d7","timestamp":1516890748.076,"data":"一番目のブロックです","hash":"745c2889d37d8a0f38f8ab02d403d628fdb525517bb4
6c2a22ee043951f5271"}]]ubuntu@ubuntu-xenial: /naivechain$
```

図 5.2 ブロックチェーンへのブロック追加

先ほど作成、追加したブロックの情報を取得する事ができる。

### 5.1.3 他人のノードからのブロックの作成とブロックチェーンへの追加

以下のコマンドを、同一ネットワーク上で繋がっている仮想マシンの実装されている他人のノードから入力した場合でもブロックチェーン上にブロックを追加する事ができる。この場合、同一ネットワーク上で繋がっている限り、naivechain のプログラムや、docker や docker compose の入っていないノードからでも追加することができる。

以下のように、naivechain の入っているノード ip アドレスを含んだ url を記入してブロックに追加する事で、追加するブロックの送り先を指定することができる。

```
curl -H "Content-type:application/json" --data
'{"data" : "鈴木のマシンから二番目のブロック"}'
http://10.100.192.78:3001/mineBlock
```

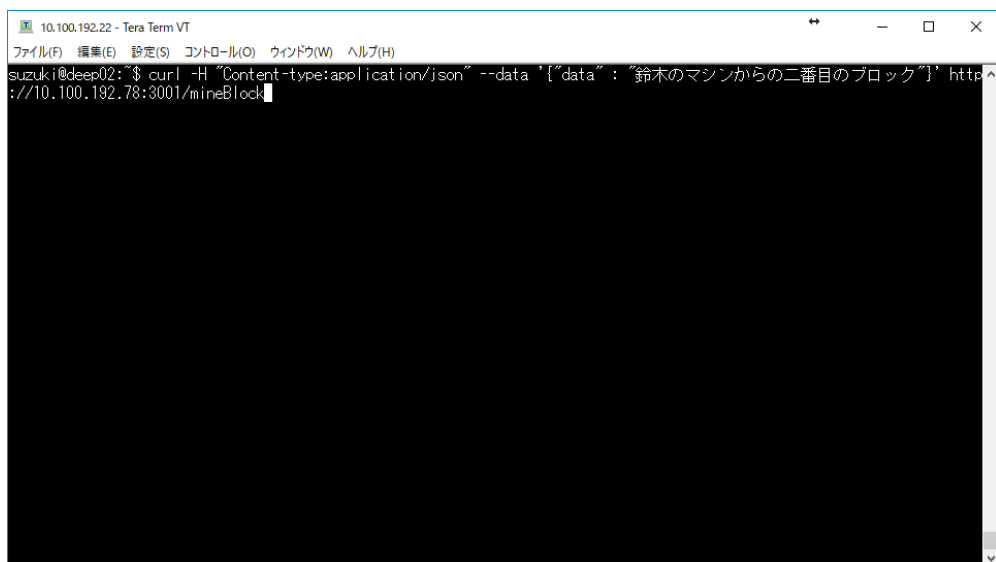



図 5.3 同一ネットワーク上でつながっている他人の仮想マシンからのブロックの作成と追加

#### 5.1.4 他人のノードから追加されたブロックの確認

ブロックの確認で行ったように、下記のコマンドを実行すれば、他ノードから書き込まれたブロックを確認することができる。

```
curl http://localhost:3001/blocks
```

A terminal window titled "管理者: コマンドプロンプト - vagrant ssh" showing the output of the command `curl http://localhost:3001/blocks`. The output is a JSON array of three blocks. The first block is the genesis block with index 0 and data "my genesis block!!". The second block has index 1 and data "一番目のブロックです". The third block has index 2 and data "鈴木のマシンからの二番目のブロック". Each block contains a timestamp, previous hash, data, and a new hash.

```
ubuntu@ubuntu-xenial: /naivechain$ curl http://localhost:3001/blocks
[[{"index":0,"previousHash":"0","timestamp":1465154705,"data":"my genesis block!!","hash":"816534932c2b7154836da6afc367695e6337db8a921823784c14378abed4f7d7"}, {"index":1,"previousHash":"816534932c2b7154836da6afc367695e6337db8a921823784c14378abed4f7d7","timestamp":1516890748.076,"data":"一番目のブロックです","hash":"745c2889d37d8a0f38f8ab02d403d628fdbc525517bb46c2a22ee043951f5271"}, {"index":2,"previousHash":"745c2889d37d8a0f38f8ab02d403d628fdbc525517bb46c2a22ee043951f5271","timestamp":1516890928.073,"data":"鈴木のマシンからの二番目のブロック","hash":"079bb0d2c4f92f7259b455dae395efac1f9af31ac878052cc54f7a2cc1d005e7"}]]ubuntu@ubuntu-xenial: /naivechain$
```

図 5.4 同一ネットワーク上でつながっている他人の仮想マシンからのブロックの作成と追加

## 5.2 Bitcoin Core

Bitcoin Core という仮想通貨のアプリケーションでローカル PC 上にテストネットワークを構築し，複数のアカウント間でビットコインを送金できた．

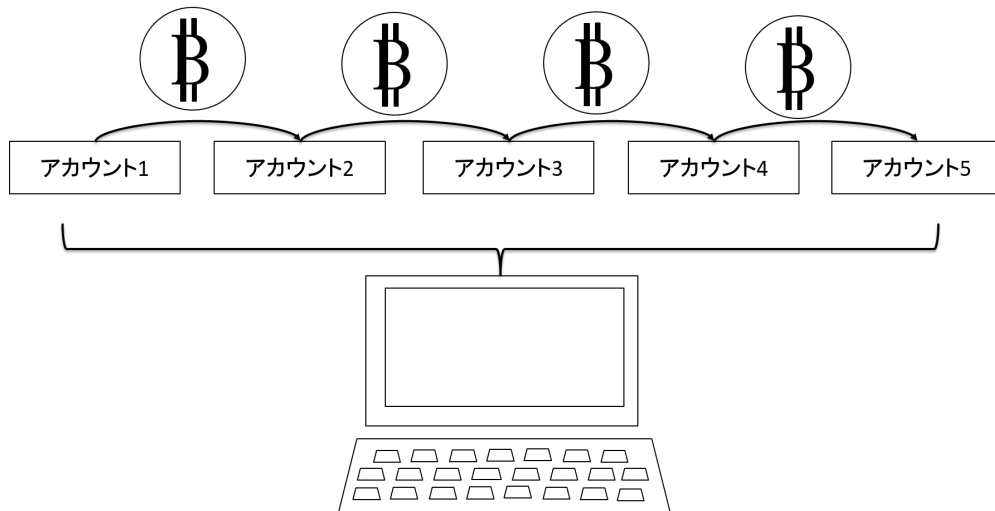


図 5.5 Bitcoin の流れ



## 5.2.1 ブロックの生成

ビットコインを送金するためには BTC が必要。ビットコインではブロックを作成する報酬として BTC を受け取ることができる。ビットコインは報酬を受け取ってから、100 ブロック経ないと送金などに利用できないため、最初に 101 個のブロックを作成した。

```
ubuntu@ubuntu-xenial:~/src/bitcoin$ bitcoin-cli -regtest generate 101  
[
```

```
"7ef25c199888f2c43295db9546167acf9ff626a84826d10072fa815205ce7d46",  
"7f35aa62a3a56378510b0f3c9fc3a7fa6275cecd73f14ab0b079e057df0b0026",  
"7278455dc0a691391c59895b214cea7aa844199c1436b9ac03a25c834472ddfd",  
"4a3f654bbeeb168dd61faf3b36ee126db166bdab198abf053b6ed24d300cddbd",  
"23a8d44a4e962c6c4beb61b4f26e30bf9ec895d66cd4677743484736128d7877",  
"3fa6a4e518056c4d4c162911623e51e4a8ab8780c0068d7226a0449bc31cba40",  
"646c7a1bd2b66c0ae1914cdf0547b46481597c61b2ddd69b50ac03c807f476b1",  
"307a32619b557d10a8b8b2f2d740b89b8b4b737d77583c5695fe56dd44980ab2",  
"560c454fa7f30142de7c9200164473a82ec09ec27a6e254a418af634b60f6158",  
"585c3a3cc87edb1857008d3c8fef6b75bccaf6b1459f632684d3658d3f100193",  
"1908ca15d23a10ea4a1a51819269a5ef1ba03612c2b960cdba797ff21beeb19c9",  
"7d523b9d897531bfe2d0fe3c40e33b1b1655b0c64d588739c7024f6cfc295da9",  
"537725b25b67e5eea1be700f95526a0a4b090573f8498976bcd90bfa055f84de",  
"79d012af4e0ef91f99e737272a106912d5c25940d93fef6e8abe7f25f0d7963f",  
"0c46b586faecbef519a7d709a9a5ca22792929833750f63536abe7b7c0951ed2",  
"346ce292b20fa9e66a40eea731b956144da4c30c34ed1ad486dab692a286962f",  
"15b162514cf2524673875bf09278f62d5ac6b8066bcdd7af8f1bf6dfd610bf56",  
"1510000dae3e601a2dcc464d03be6f5eec7fac4e55ea45e657c3c912db634c25",  
"7988656001fe2d65feebe8cb9759bae3f67f8925c609238008aa292f033f4da8",  
"27f8116b51859ebbf3eb99d16eb1d9daa3f4f84f2c05b8e24048da2a8f8d9e2",  
"5695233d0603d7b7e3a024f95ccf37b385410e66b3d69fb2a27726299e96e098",  
"575b40c7a6bed5f87689bc278bbffa5450b4a6288b44d9bfb63a66c7ac35e4e3",  
"5c79289002844fef24d560ad1477a0f25b3faea94c9c9db421d615608d44f7f5",  
"3fd93a7637031e6835f9e0458642845f2ba06d862ecbc41da2aa459141140635",  
"4cc549d80c0ba36b063b5eaa4301a74e8fdf871894240347263cb84b96a910b7",  
"63e6fd42ac0c3be1e90e4290f0125cd15d0133ddd40eb77a2b35590f777e1443",  
"612eae3b841c1bf6738591207c3831a9f14ee97dc37d3a5dcdbc68a1a4c6c55",  
"2c8abbfead60aa1691461d3dcf3a4e13826891043c9710d62e9ef9a99906fac6",  
"5eb510455813c8815dd02348a7605182df3877e8d84314da671bafdb2b2f7c1ed",  
"67f3fe4032165bd02cc143ba409008541e62898af3e0b02e17704b88f0fadefa",  
"1719ada0cc87a4161d0c335f06f5ce754447f650bbb81b56bdad3c875834594c",
```

"0ec057a6f1ecbf455acbfe92c18f183a4c3c9c5e52d6524429e39f449971f649",  
"7bb3dc4bc776555258965c4a09cb0227ff51c0cb6f936d3ae2c404df08aba7e4",  
"2ac58ef71bdb7aa24367d03f74a50a3d0bef15b5d36e14d05809702bf870cc61",  
"6b21e97a65fbce0fa3859ad89a9d6d54dca3d0036000c190e205c6a34f0f6d8c",  
"42d902d7cf3d8eead40173c7eb97ae331f414e8ce1e31c3431c6f115eeb30a45",  
"2657af8e06a88e8ccbd4d9db0a01646a49ddb02876441a75e2d508ba8435f74",  
"7030cae00f73023a9cd28af5a3775c96c064a9151e7d354af8a38c5800b64540",  
"24f49562f97c3fb6ac7959fee4929c9365c97d9b508464d6c19834e06a8782e1",  
"2b056aa1f8839c1fe043c05375164538536dc910b967d10400a1fa3f247e0a36",  
"184652bb56b38dd4687d5c4434d044d967da05412acb0ca1744cd1bdf5bd69eb",  
"4e18a73c050567239fb917a4b62cbdb0a9e98b911485874890bf456c52e4d83d",  
"55d05176368c4f7fed472bcffcd1938d0dae4dd90d285f02562d9073c14ddf1",  
"2d594305f00caa36fdef6167db18689abc5ef3fd94eaff3316e7dab92fb0c472",  
"397648d50807ffebbbb34efb6f1c1301feba6d7d736c88894cc96ce4f951613a",  
"1886fc9bed45adbdd55baf62af14c49ceab460044d2fe9751ddbfb17f59ea884",  
"2268fb65e8a999dd7ca741430614f985bd68e8cae70ae9259107f5142e28d262",  
"234a4961df925d5ffe18b7c6c8121e7ede295a146a0059cab182cd0fe929c285",  
"03b94bf0815b915a98909661d5c6e4a11d37248bede296620dcabb670904b5b2",  
"5c66bd8450743e5da1b5788e1b2ecea6260cd1e9c10e42e41fd19ea4c93d2960",  
"0c49ea0eedb86972290b38137be15f07fc26fcaaed0c173304628759cabdd04c",  
"6be1381e4dc94c3b0a96d46cb9bbd6a20cc0b73e25e3eabbe2ec7931a57133b5",  
"3bb60d0e12a48d98dbbc9d1692aa50972338dc6c95e4dc0515a021f6667e2fe9",  
"2d31ecde9f6771ce638a4afad5fb0b76e0955eadec4af4acb2446b68f9f10a54",  
"46c15a821ec1f78e24e0edc6387c844912d9b13bbe195c052a22fdd0e139f81d",  
"48030cbee97c2824768bbf0c793b6297d6c78007f372ad4fe0a85b08f60f9167",  
"68cda2593cddc4389dc6799d1dad97efc853c66ad28ba59bc77b4b38797246b0",  
"6a1ffab708d366ee2b33b173db82b5f9112251e165829bce9af4ae20f761520d",  
"120da86717ee55eafbfb10439b5fa6a06be8b48d298aafd5d65d48810dfccb53",  
"5f0ee6fff7a84d5bac7af917d7186d21b582ee79641016743a31c2a841144d10e",  
"1f8d24d4e7303928d7a5a8c5992607fa67ffab4c61e0f939ac190eebf1d28194",  
"6b7a8abfb5d46aad2daf842a1d3de0c51683b4a36af349d71dc28b899f01994c",  
"19cc387a183497c034e9156238e8dea2055dd282bab35c4f40088b8be00f563c",  
"678f3aabe7c0587a963cd3deea26ce19b55cf7fa8efb1ad40129b402add5854e",  
"2334ea82c178a4af7fcc4cf34de4d0091995ba927743b3ec6ed04b2543399dde",  
"7ce23f972b108e711d973fdb984cff34748fced371843bccd31e7ac1cfcb577c",  
"1041e3c85623d6b2fcda515a1df8ef6c9280e9c434e446dc1d1005c10944ca99",  
"0a43ad6b6cbae9bbfc1d61dc3b2099fe45229804b4089f3032c5f39ba75427c1",  
"176e12f0adc1ce42d4f3aee818e8e31cad463fe73e87044e598f4c4733aa0568",  
"15d66bf6a71ff8b4596b870416a34073d7160dccd391621350e2bdcc6f82813b",  
"377602127294306c3f0e483e20122cf7a05138e0dd549c2aff682eafdaa2ad6c",

"1e1e9c8dbc5aeab49d988aa867493558272efcada6585dcd927dac8c3ae416b9",  
"168c5b863835559e97a31ffcbbf16ce5bf8ef0088544bc96b4dc5822c5948ee0",  
"7febdbe9e2c4b2ee878af1cd64553ed1f3307ac82b21c2477a030e2d36503dc8",  
"589c02a454cf81a09d288416c215f811d7523efdf6f3f84962ba702958f72f54",  
"2a7082909cad6c86bcff1d9b0d31014c085f92723ca86349fa48c41eba8a663a",  
"072562cd981196cedd917e8f0a0d3f6a1a535e1a7a487e9b983c1f772e2c2bbb",  
"43d82e8d78ba89d8c44e767f1b7f0b36bfca24cde8e94a1709319d9f8bf55085",  
"51442ccd3796582aefeaccfa1f865df71377f10d9be93778377d919b5cc4bca7",  
"74ae9898c5fa5ae0d812184dc253a0fd0f3499f933a246b877011665f2924ee0",  
"7e8045afc5d4672d09aea83f3231810ba4104f804e35f983ae463dbed3f813a0",  
"2c3769e27e93cf9297eb1cd1c4f7bc46f0714431d053d21a805071565cc5de1b",  
"5a66e5ed427fd5157c811bef29a0edb1c0503c9ec86951a6fc1767841204b478",  
"1927cfc29f42169af7258b7f1c0b3a97aed3fc741928f764329a974e649c656b",  
"2b120249c13ee892e711cfd7e086f3862c7c0c1d6bf0b9570071f79c68ca950",  
"24fba31ad43d7e35b0c0e0474b88ab9f801a4fec84a8b40c4688f2d810e8f0f3",  
"0c4ab3e8dc4f9b765130bf170b127c291261cac5ddc89ccb0e83fc8a90d3546a",  
"1932cb1a400de7ba40e3d216ca69133dc18deefe447fa564200cfa8631f24077",  
"0045cae03c10d6c7e917f68042976d96096ee0ce0ed2dd807a5c66400053b52d",  
"099c16751261f8584c471cb4cf7f377d8ec2cec57258fd79e94036610390f6af",  
"5caab5807a08380f1d6aabc34572f577ef03ac4298d4468bd34aa1cbe75345d9",  
"4326308d69fedce5bcfec688fbf26b1793614723fec4947f08324e346d75a824",  
"33a700fc422c64ec87fd42c649e3f6fcc9a0fe58605f3fa4dadbfa223c0af943",  
"4122cc92994fe22ab45786334bd025a91e8cdc0123026e0b5a55af6f9fdeb358",  
"007e581ee8406f30f223bd4048b817eb654bfcf376424f4b1d499165e8584359",  
"1b15dc5d48d9b47fe875946607e941fc98de0607b8bd2533a20921c18a8e321a",  
"42fc0d009994d2f98fa2a4206fefde219800d4e4252338cf897f4cc0ac2a98db",  
"4370bb3459050f1e7bc9074c3a2f0718be6d3da0c5231657476696cf08dcf101",  
"32dfeeee098e3db96460e7c5fccbe3b0e38fa5afc7d8464a5d287a14a61ba307",  
"136824554867cf7c86238a3b6d8b9168c7fb7a3454d76e35bb6e955c3bfb05d2",  
"7b05d0f07d737f47166ce21dcfae7646cf4e50d16dfc0b1176af9231e5f17e4a"

]

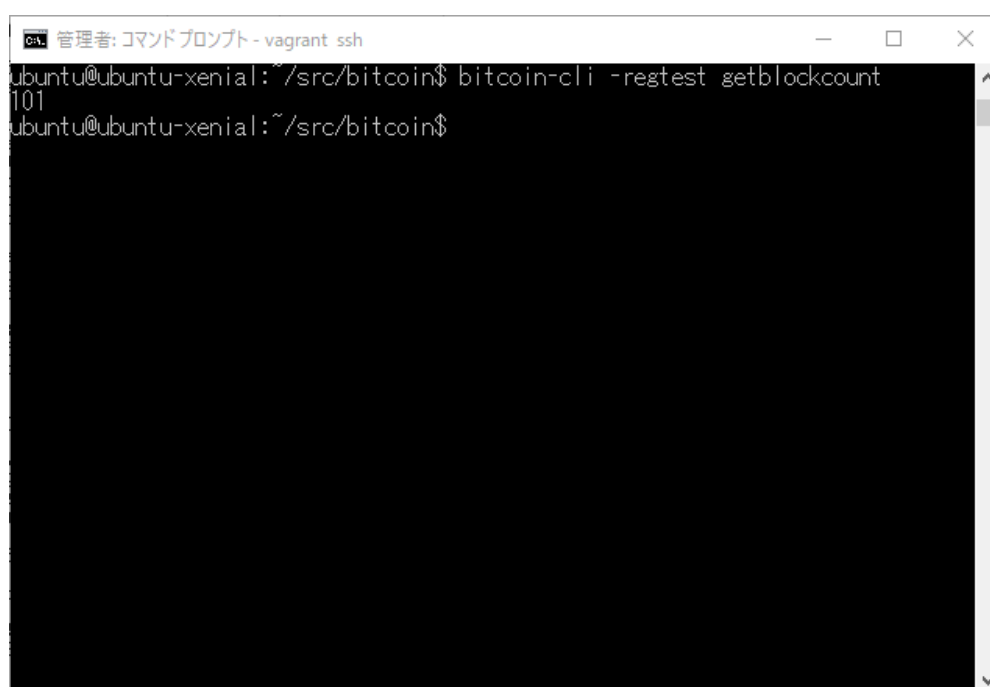
### 5.2.2 ブロック数の確認

101 個のブロックを作成しているので結果は 101 になる.

```
bitcoin-cli -regtest getblockcount
```

結果:

101

A screenshot of a terminal window titled "管理者: コマンドプロンプト - vagrant ssh". The terminal shows the command "bitcoin-cli -regtest getblockcount" being executed, and the output "101" is displayed on the next line. The prompt "ubuntu@ubuntu-xenial:~/src/bitcoin\$" is visible before and after the command.

```
ubuntu@ubuntu-xenial:~/src/bitcoin$ bitcoin-cli -regtest getblockcount
101
ubuntu@ubuntu-xenial:~/src/bitcoin$
```

図 5.6 ブロック数の確認

### 5.2.3 アカウントの生成

アカウントを生成する。銀行における口座に相当する。この手順によって口座間の BTC のやりとりが可能になる。アカウントは何個でも生成できる。

```
bitcoin-cli -regtest getnewaddress testuser2
```

結果:

```
mwXKjNW7quodFpoTMnPTwtqCGZ2d9v41KY
```

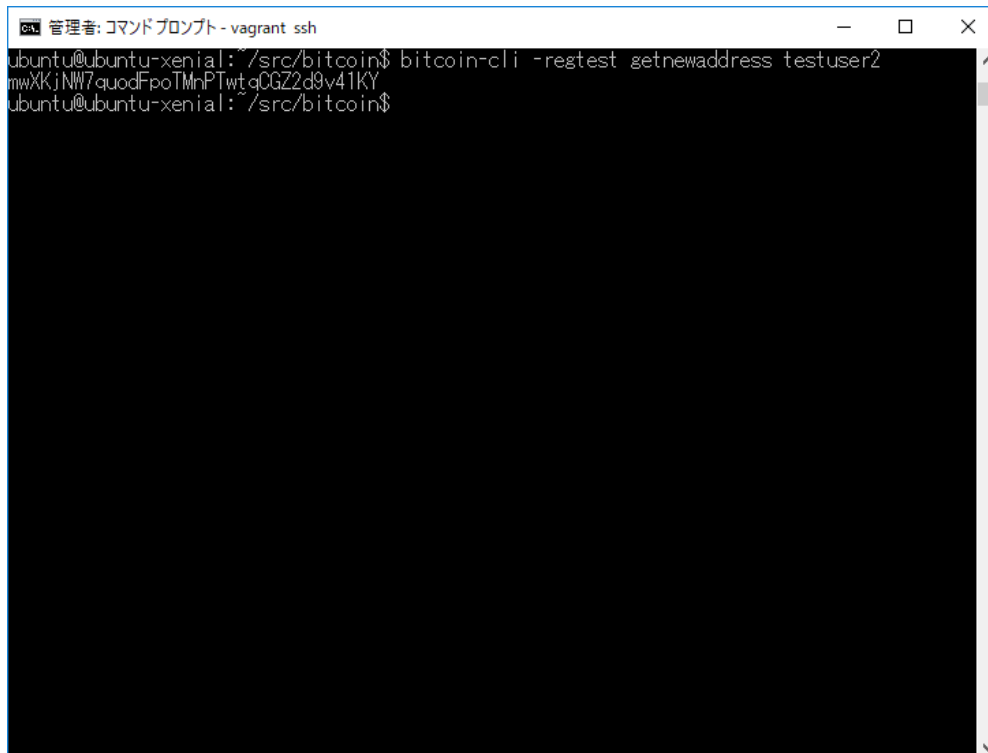
A screenshot of a terminal window titled "管理者: コマンドプロンプト - vagrant ssh". The terminal shows a user at the prompt "ubuntu@ubuntu-xenial:~/src/bitcoin\$" typing the command "bitcoin-cli -regtest getnewaddress testuser2". The output of the command is "mwXKjNW7quodFpoTMnPTwtqCGZ2d9v41KY". The terminal window has a standard Linux-style title bar with minimize, maximize, and close buttons.

図 5.7 アカウントの生成

## 5.2.4 残高の確認

送金処理を行う前に現在の残高を確認する。50BTC が口座に存在している。引数なしでコマンドを実行した場合、マイナーが所有する BTC が表示される。

```
bitcoin-cli -regtest getbalance
```

結果:

50.00000000

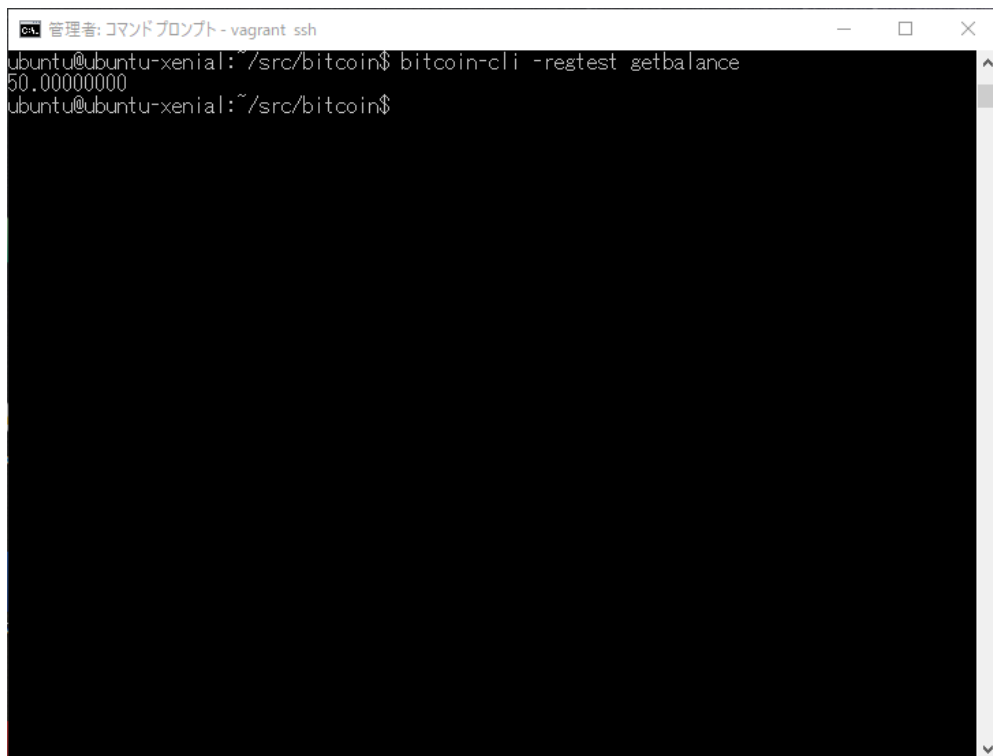


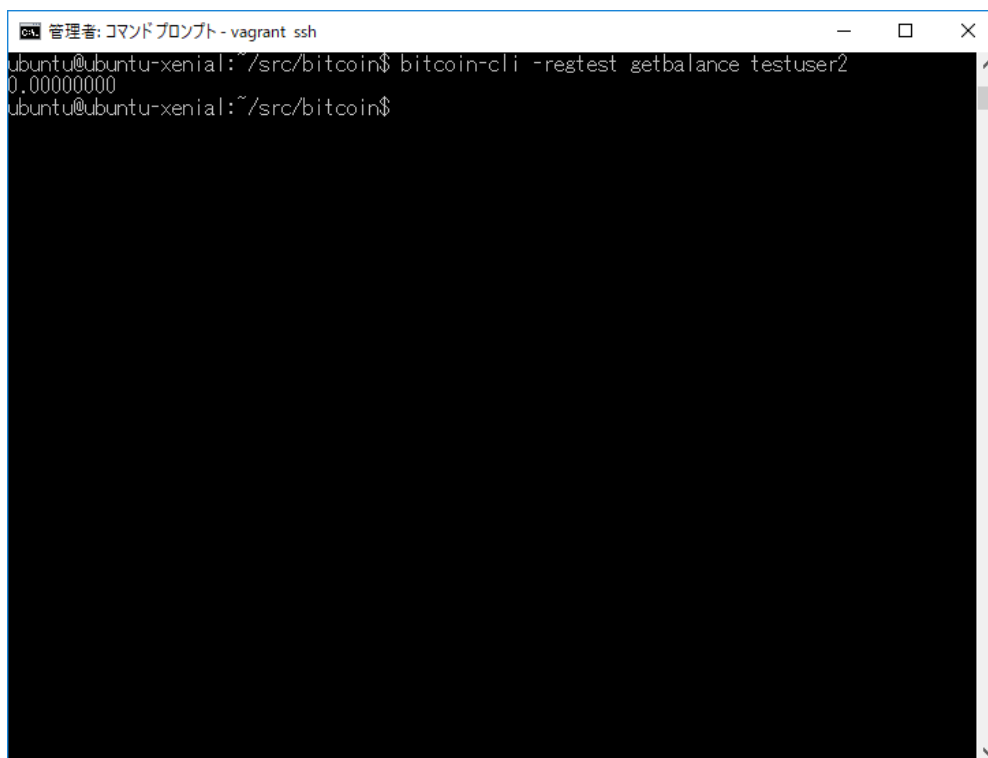
図 5.8 残高の確認

特定アカウントの残高確認。先に作成したアカウントの残高を確認する。testuser2 のアカウントに BTC は入っていない。

```
bitcoin-cli -regtest getbalance testuser2
```

結果:

0.00000000



A terminal window titled "管理者: コマンドプロンプト - vagrant ssh" is shown. The prompt is "ubuntu@ubuntu-xenial:~/src/bitcoin\$". The command "bitcoin-cli -regtest getbalance testuser2" is entered, and the output "0.00000000" is displayed. The prompt then changes to "ubuntu@ubuntu-xenial:~/src/bitcoin\$".

```
管理者: コマンドプロンプト - vagrant ssh
ubuntu@ubuntu-xenial:~/src/bitcoin$ bitcoin-cli -regtest getbalance testuser2
0.00000000
ubuntu@ubuntu-xenial:~/src/bitcoin$
```

図 5.9 特定アカウントの残高確認

## 5.2.5 送金

作成したアカウントに実際に送金する。送金処理を行うためには送金先、送金額を指定してトランザクションを発行する必要がある。10BTC を testuser2 に送金した結果を掲載する。結果として表示されたのは、トランザクションを特定するための識別番号 (txid)。

```
bitcoin-cli -regtest sendtoaddress mwXKjNW7quodFpoTMnPTwtqCGZ2d9v41KY 10
```

結果:

```
2e45d07dcfb191e0d4f6ca4d1f302600a0e94b02668788ec7fbdcbbba0fe723cb
```

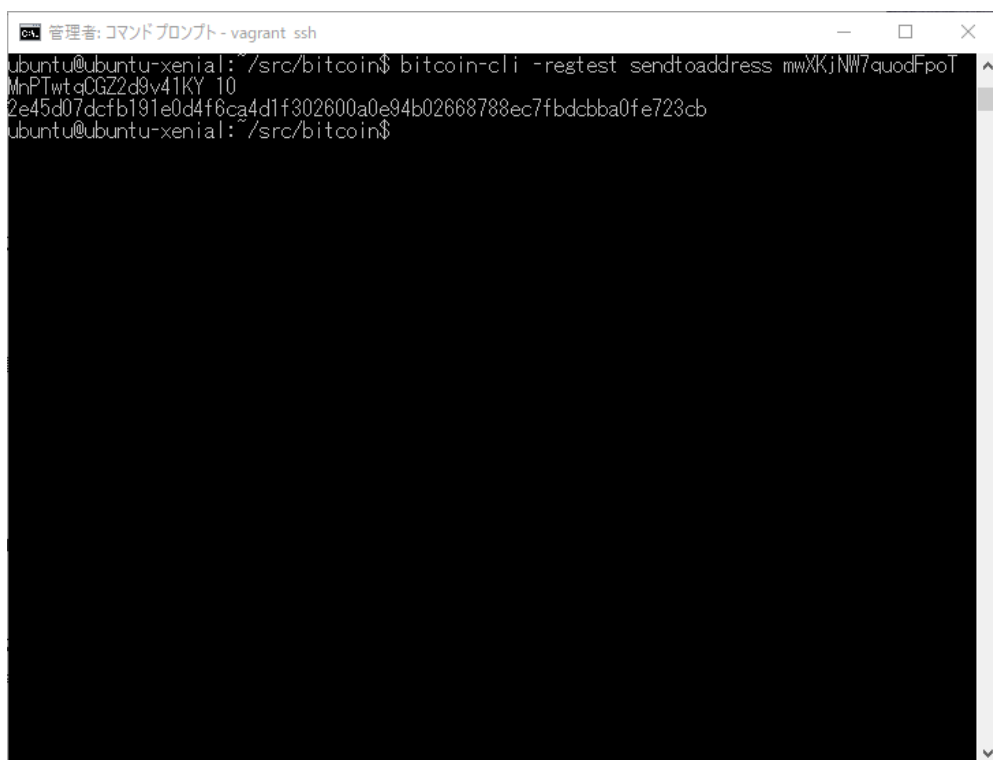


図 5.10 送金

出力結果は空になる。listunspent は確定したトランザクションを確認するコマンド。

```
bitcoin-cli -regtest listunspent
```

結果:

```
[  
]
```



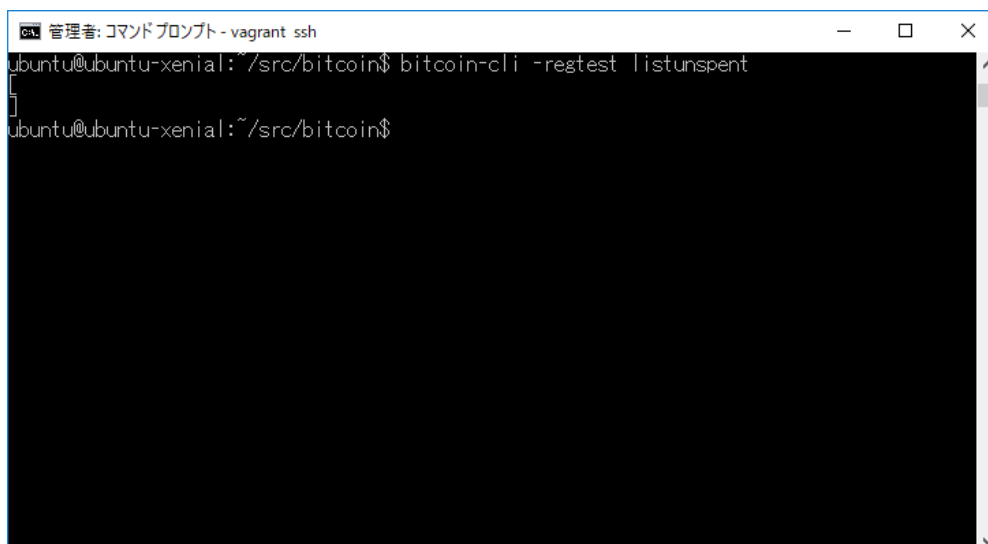


図 5.11 トランザクションの確認

```
bitcoin-cli -regtest listunspent 0
```

結果:

```
[
  {
    "txid": "2e45d07dcfb191e0d4f6ca4d1f302600a0e94b02668788ec7fbdcbbba0fe723cb",
    "vout": 0,
    "address": "mu9gChKwwcmX1uHwRsXM2qKBS9s615T3Pf",
    "scriptPubKey": "76a914958b97b05b8e3d11ce3ec5cde8fef92a05d0fbf188ac",
    "amount": 39.99996160,
    "confirmations": 0,
    "spendable": true,
    "solvable": true,
    "safe": true
  },

```

\newpage

```
{
  "txid": "2e45d07dcfb191e0d4f6ca4d1f302600a0e94b02668788ec7fbdcbbba0fe723cb",
  "vout": 1,
  "address": "mwXKjNW7quodFpoTMnPTwtqCGZ2d9v41KY",
  "account": "testuser2",
  "scriptPubKey": "76a914af93e82677e868782ae9a260f2eea64a2014783688ac",
  "amount": 10.00000000,

```

```
    "confirmations": 0,  
    "spendable": true,  
    "solvable": true,  
    "safe": true  
  }  
]
```

先ほど表示されたトランザクションの識別番号と同じ値で、14 行目の"address"に送金先アカウント、17 行目に送金額が出力される。また 5 行目の"address"が送金元アカウントであり、7 行目の"amount"に送金後の送金元アカウントの金額が表示されている。

## 5.2.6 マイニング

未確定のトランザクションを確定するためにマイニングを実行した。ブロックチェーンではマイニングによってトランザクションがブロックに書き込まれることで送金が確定する。

```
bitcoin-cli -regtest generate 1
```

結果:

```
[  
  "16d9d752607d47d67a944a6bb7d04178e598407c07b187e34b6238868261a970"  
]
```

## 5.2.7 送金の確認

```
bitcoin-cli -regtest getbalance testuser2
```

結果:

```
10.00000000
```

アカウントに送金額の 10BTC が存在していることがわかる。送金に成功した。

## 第 6 章

### 考察

データの改ざんが困難なブロックチェーン上で誰もが記録を閲覧できる本研究は、Akatsuki のようなソーシャルゲームサービスを提供している会社の意図的な不正と、ユーザー側の一方的な誤解を防ぐ証明として役立つのではないかと考えられる。しかしユーザーが意図的に誤った情報をブロックチェーンに書き込んでしまう可能性もあるので、ブロックチェーンにブロックを追加する際には有料アイテムの抽選結果をユーザーによって改変できない仕組みが必要である。

## 第 7 章

### 結論

以上の結果から，ブロックチェーンに記録されたゲーム内乱数の信憑性は高いと言える．  
今後は抽選結果をコンピュータが自動的に判断してブロックチェーンに追加するシステム  
を実装することが今後の課題である．

## 参考文献

- [1] 広田望. ブロックチェーン (Blockchain). 日本経済新聞社, 2016.
- [2] 丸山和子, 愛敬真生. 文系でもわかるブロックチェーン. 日経 BP 社, 2017.
- [3] wikipedia. Docker. <https://ja.wikipedia.org/wiki/Docker> (2018.1.25 閲覧) .
- [4] 松島浩道. 複数の docker コンテナを自動で立ち上げる構成管理ツール「docker compose」  
(docker の最新機能を使ってみよう:第 7 回). <https://knowledge.sakura.ad.jp/5736/> (2018.1.25 閲覧) .
- [5] 赤羽喜治, 愛敬真生. ブロックチェーン 仕組みと倫理. リックテレコム社, 2016.

# 謝辞

本研究を進めるにあたり，ご指導を頂いた卒業論文指導教員の矢吹太朗准教授に感謝致します．先生には研究以外にも多くのことを教えていただきました．研究の進め方や論文の書き方など，ひとかたならぬご指導をいただきました．言葉を尽くしても足りないほど感謝しております．本当にありがとうございました．また、研究や日常の議論を通じて多くの発見をさせてくれた矢吹研究室の皆様，その他本研究や大学生活全体を通して関わっていただいた皆様にも多大なる感謝の意をここに表します．