Linuxサーバを作って壊そう (配布用)

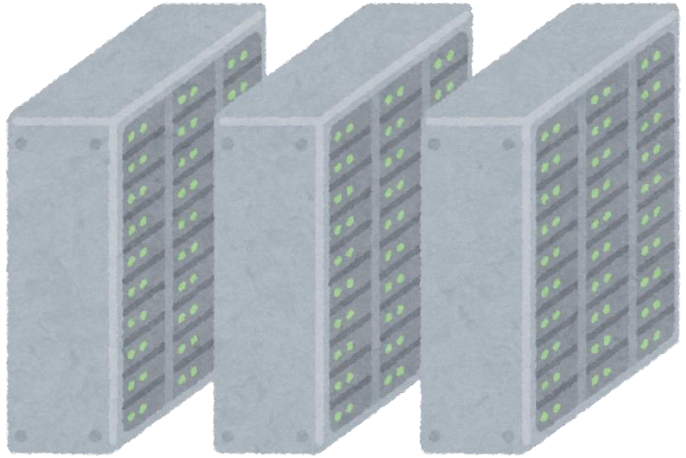
AICサーバ班 メンバー

竹内奏人　杉山晴　沼澤貴大

原駿平　岩崎一樹

AICサーバ班

- AIC JupyterHubサービスの構築・運用
- 新たなGPU利用サービスの構築



前提

- 本講習は中級講習
- 前提：

  基本的なLinux CLI操作（ls, cd, mv, rmなど）ができる

- 復習はもちろん行います！
- TAにも聞いてください！

おやくそく

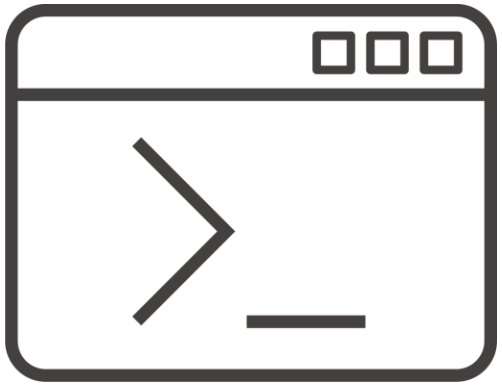
- 本講習ではグループでの助け合いを推奨しています
- LLMの使用も推奨しています
- 講習中でもいっぱい聞きましょう

注意

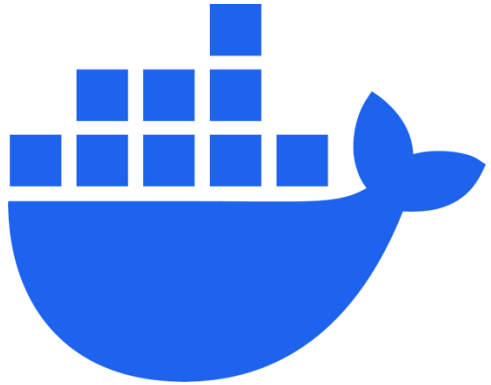
- この講習は初めての試みです
- X(Twitter)でバズり散らかしていましたが、
  至らない点も多いかと思います
- 今後のためにもぜひフィードバックをお願いします

本日の流れ

概論
「大学のサバ管」



Linux管理者のための
基本操作



Webサーバ
on Docker

トラブルシューティング

破壊

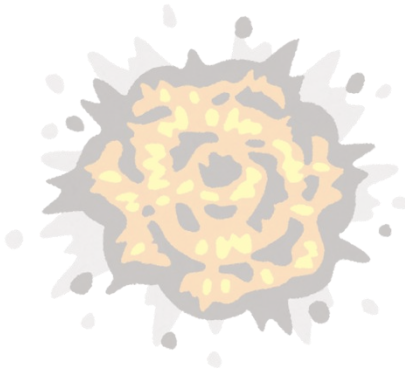
本日の流れ

概論
「大学のサバ管」

Linux管理者のための
基本操作

Webサーバ
on Docker

トラブルシューティング



破壊

概論「大学のサバ管」

竹内奏人

概論：SABAKAN

鯖カン

概論：SABAKAN

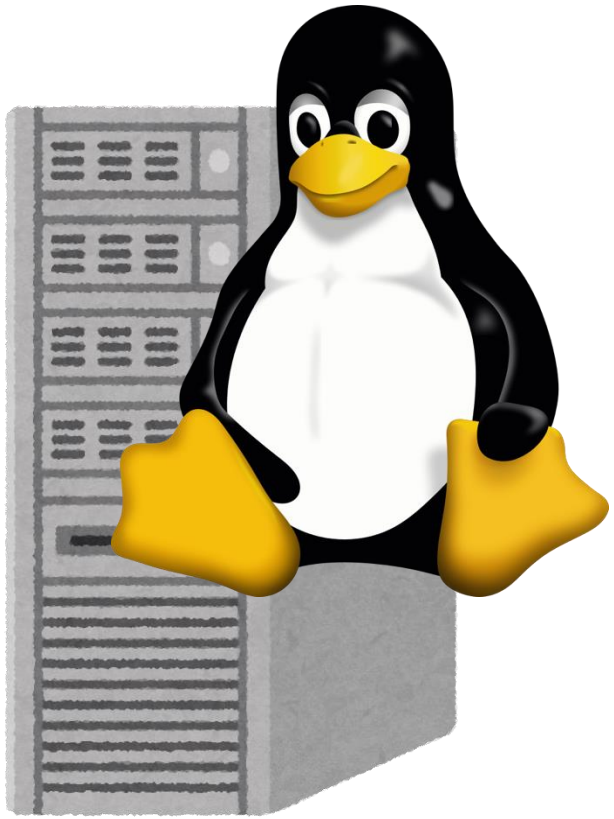
サバ管(サーバー管理)

ユーザの要求を受け取り、それに対するサービスを提供するコンピュータやソフト

例: YouTubeのサーバ

概論：Linuxサーバ

LinuxというOSの上で構築されたサーバ
全世界のサーバの90%以上がLinuxサーバ

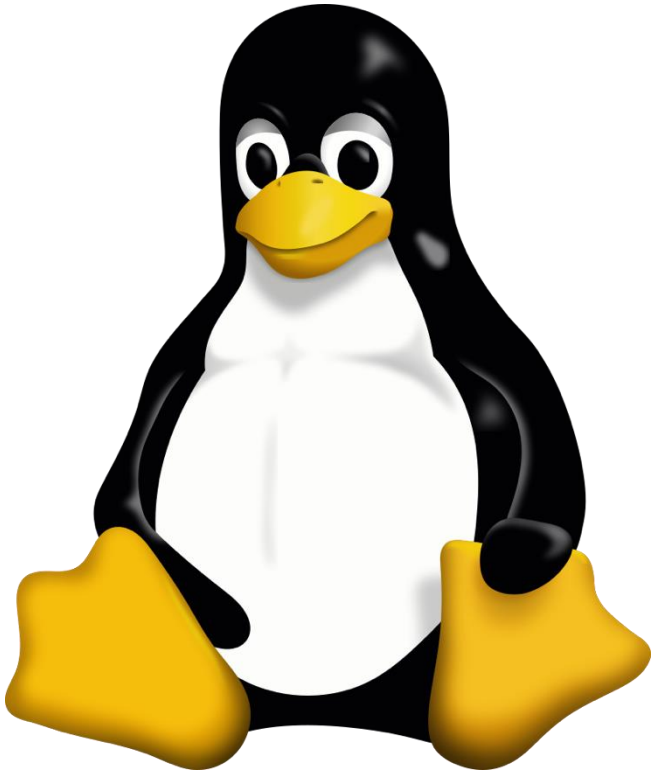


よく知るサービスもLinuxで動いている
- Googleのサービス(検索, Gmail, YouTubeなど)
- Amazon ECサイト
- Wikipedia
- X(Twitter)
- Facebook

概論：Linuxとは?

OS(Windows, macOSなど)の一種
サーバのosとして良く使われる

以下のような利点がある
- 無償で利用できる
- カスタマイズが容易
- インターネット上でノウハウが多く蓄積されている



研究室では様々なサーバが動いている

種類	内容
ファイルサーバ	実験データの置き場所を提供
Webサーバ	研究室webページを提供
GPU等計算サーバ	計算リソースを提供
バックアップサーバ	実験結果をバックアップ

サーバを有効に使えると研究で有利

概論：学術系ネットワークは狙われやすい！

大学などの学術系ネットワークは攻撃者からとても狙われやすい！

- 重要なデータ、膨大な計算リソースが多く存在する

それにもかかわらず
- 専門家でない人が管理をしていることが多く、管理が甘い
- 古いシステムや機材がそのまま使われている

大切な実験データ、計算リソースを使われないように
サーバ管理は必要不可欠

概論：サーバ運用に必要な知識

サーバ運用には多くの知識が必要

- Linuxの知識
- ネットワークの知識
- セキュリティの知識
- その他使用用途に応じた知識
  - ストレージの知識
  - HPC(High Performance Computer)の知識
  - GPUの知識
  - など…

例：某研究室で起きたセキュリティアクシデント

サーバ疎通テストのため仮のユーザを作成

ユーザ名 : test
パスワード : test

テスト後このユーザの消去を忘れた

例：某研究室で起きたセキュリティアクシデント

攻撃者がこのtestユーザを用いて不正ログイン

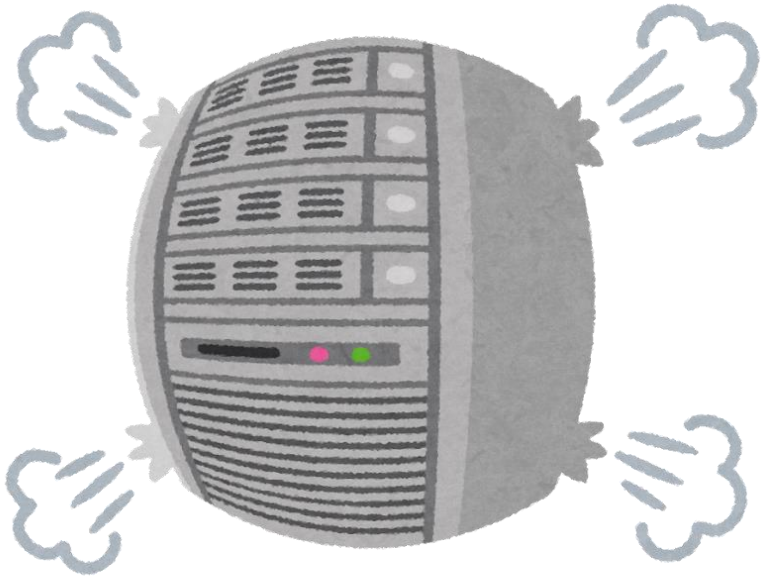
管理者権限は奪取されていなかった
仮想通貨のマイニングが行われた形跡
(クリプトジャッキング)

今日の目標

- サーバ管理に必要な知識について
  実際にLinuxを触りながら学ぶ
- 実際にサーバのトラブルを直すことで
  トラブルシューティングの勘所をつかむ

サーバ構築、管理で必要な知識を学んで
安全に活用できるようにしましょう！

存分に環境をいじってください

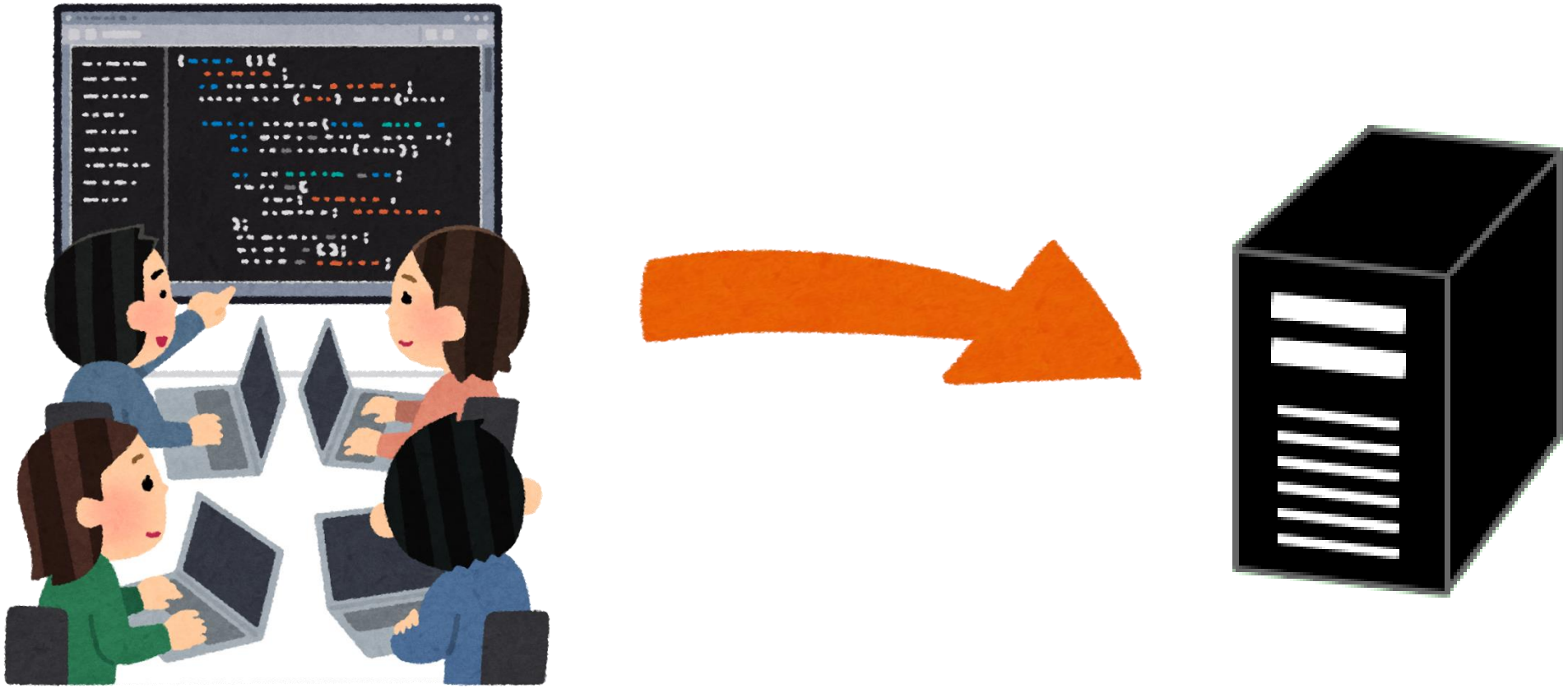


本日入っていただいている環境は仮想マシン
再起不能状態になってもすぐにリセット可能

講習内容以外の内容もぜひいろいろいじってみてください

準備：sshでサーバにログイン

サーバに接続してみましょう

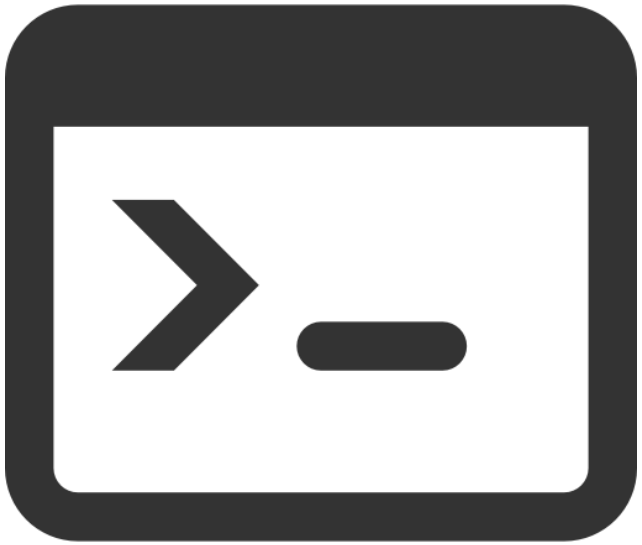


sshというものを使ってサーバに接続します

ssh(Secure Shell)

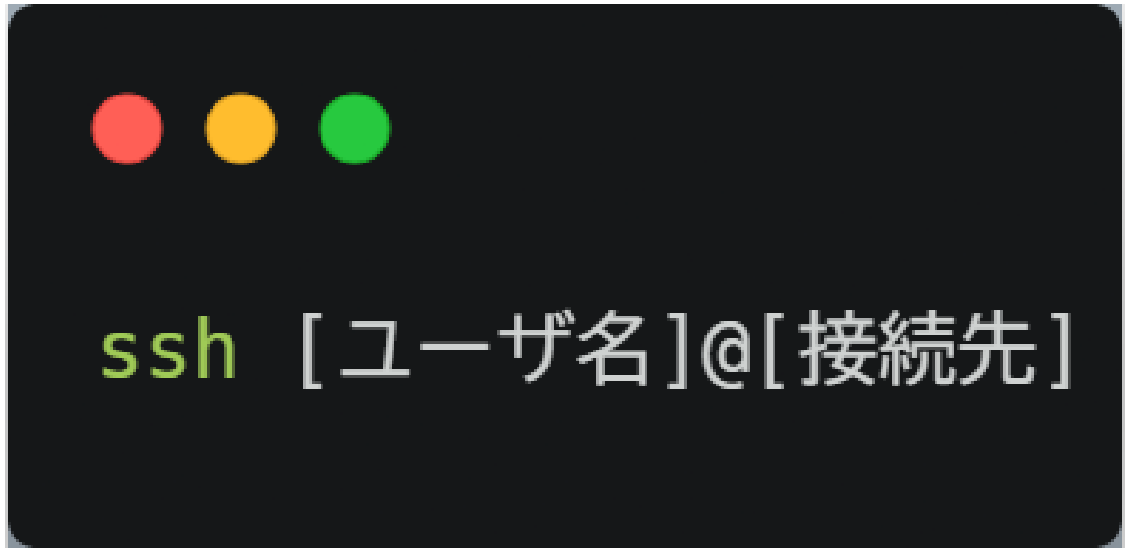
ネットワークを介して安全にコンピュータに
接続するための仕組み

サーバへの接続



今日使うサーバに接続します
各自ターミナルを開いてください

sshコマンド



ssh接続を行う際に使うコマンド
ユーザ名と接続先を指定する

公開鍵の登録

```
aic-student@root-lecture:~$
```

接続に成功!

```
exit
```

exitコマンドでサーバから退出できる

パスワード認証は危ない

ログイン時パスワード認証はあまり使うべきではない

- パスワードを忘れてしまうとログイン不可に
- パスワード漏洩による不正ログイン

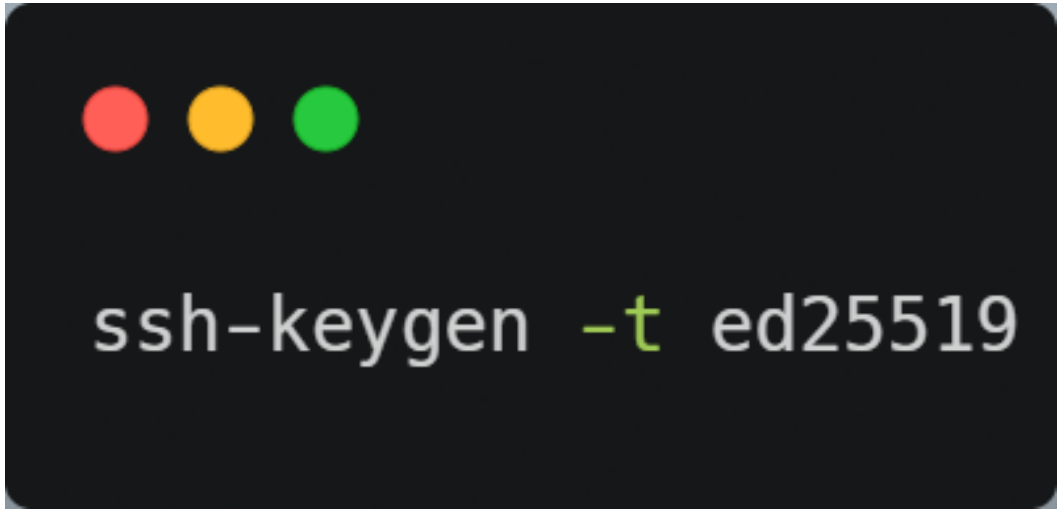
ログインには**公開鍵認証**を使いましょう

公開鍵認証とは

公開鍵と秘密鍵 2つの鍵を使って行う認証方式

事前に自分の公開鍵をサーバに登録しておくことで
パスワード入力なしでログインができる

公開鍵を登録しよう



鍵ペア未作成の方は
ssh-keygenコマンドで公開鍵を作成する
-tで鍵の方式を指定する
今回はED25519(楕円曲線暗号の一つ)を指定

公開鍵を登録しよう

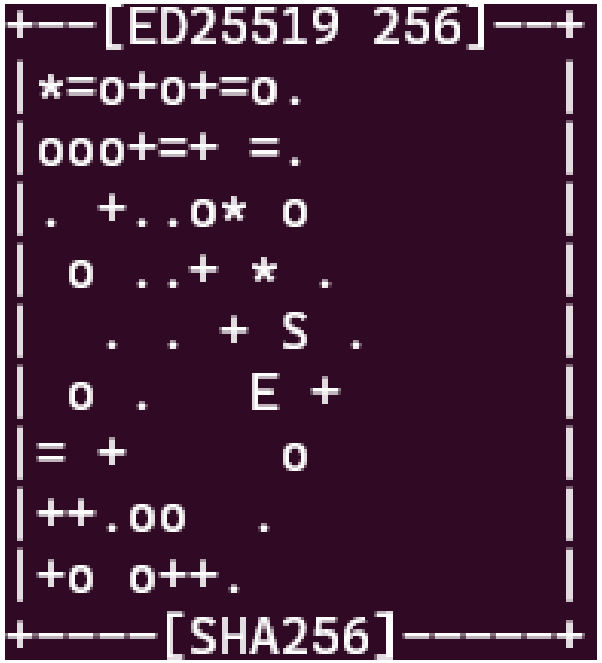
```
→  ~ ssh-keygen -t ed25519
Generating public/private ed25519 key pair.
Enter file in which to save the key (/home/kanade/.ssh/id_ed25519):
```

ファイルの場所を指定 ()内はデフォルト設定

```
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
```

鍵パスワードを設定

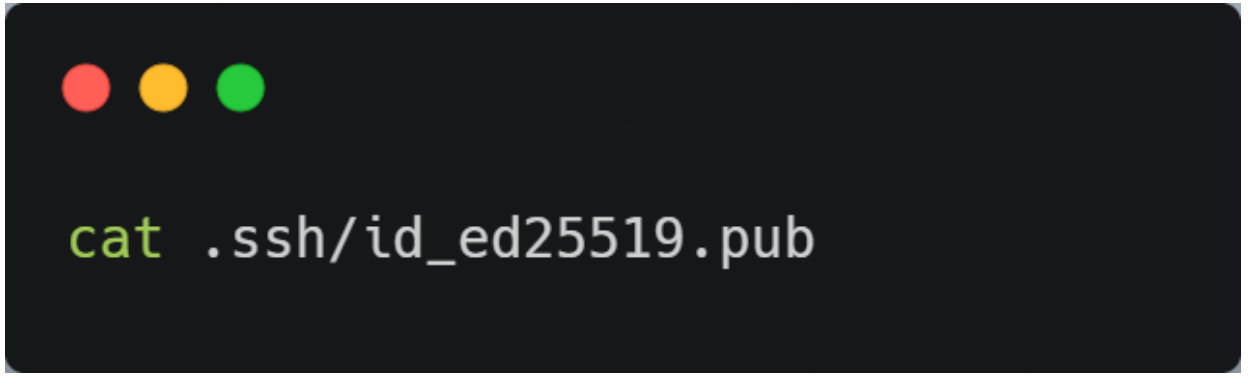
公開鍵を登録しよう



こういうのが出ればOK

公開鍵を登録しよう

catコマンドで公開鍵を表示する



.pubがついている鍵は公開鍵
ついてないのが秘密鍵

秘密鍵は漏洩厳禁
取扱注意

公開鍵を登録しよう

再度サーバに接続してから以下コマンドで公開鍵登録

```
echo "ssh-ed25519 AAAAC3NzaC..." >> .ssh/authorized_keys
```

.ssh/authorized_keysというファイルに公開鍵をコピーすることで公開鍵認証ができるようになる

再度入りなおしてパスワード入力なしで入れることを確認

パスワード認証をオフにするには…

/etc/ssh/sshd_configという設定ファイルで

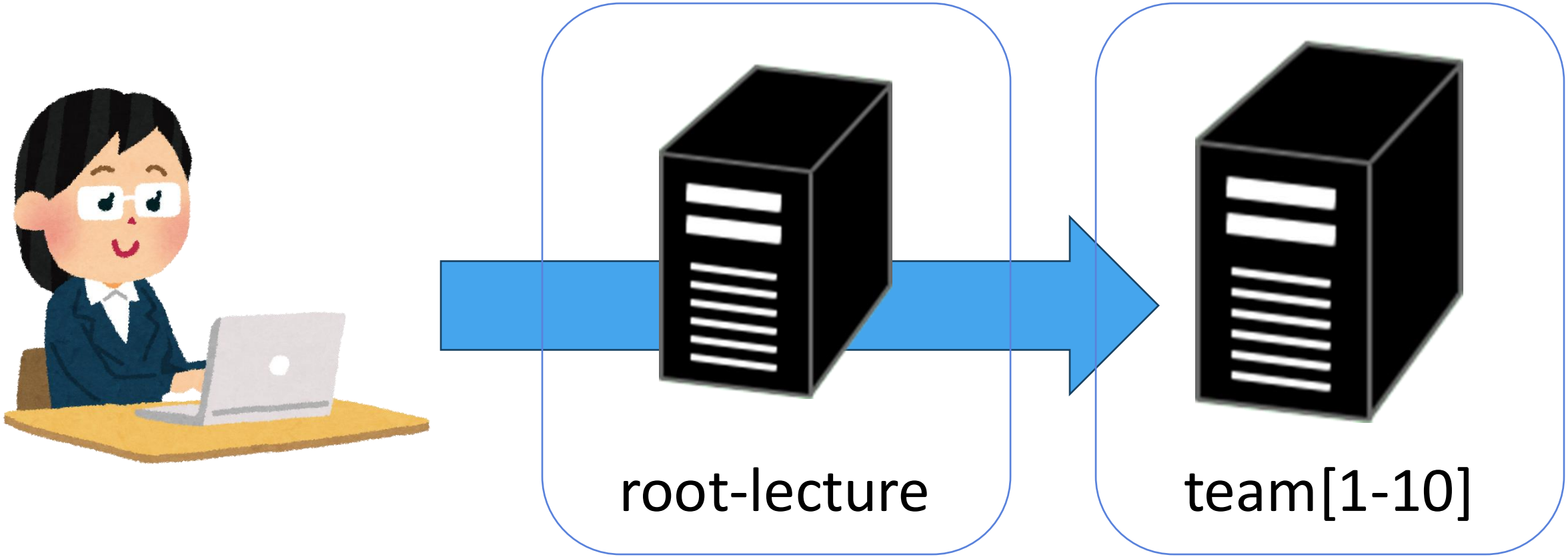
```
PasswordAuthentication no
```

という設定を行うとパスワード認証をオフにできる

公開鍵認証によるログインのみを用い
パスワード認証はオフにするのが推奨設定

本日は演習内容の影響でパスワード認証は**オフにしない**

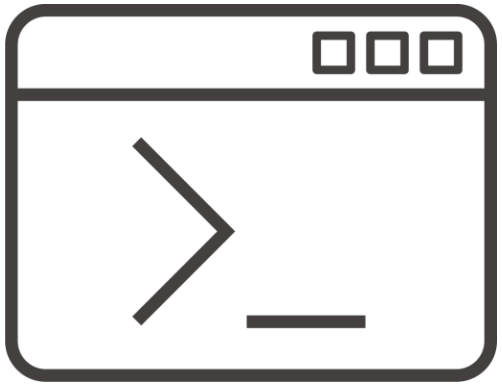
チームのサーバにログインしよう



今入ったサーバを踏み台にして各サーバにログインします

本日の流れ

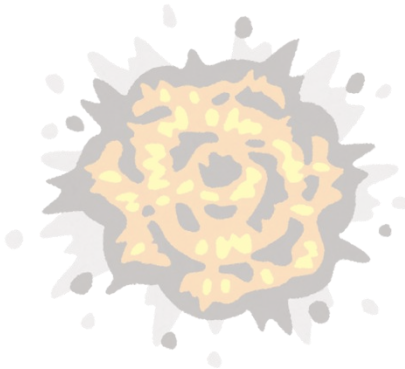
概論
「大学のサバ管」



Linux管理者のための
基本操作

Webサーバ
on Docker

トラブルシューティング



破壊

Linux管理者のための基本操作

杉山 晴・原 駿平

Linuxの管理者がすること

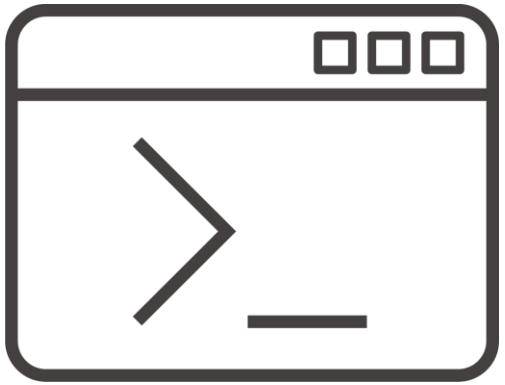
- ユーザーを作ってグループに入れる
- 権限を設定する
- ソフトウェアを使えるようにする
- 再起動後にプログラムを自動で起動させる
- 固まったプログラムに対処する

そのためには……

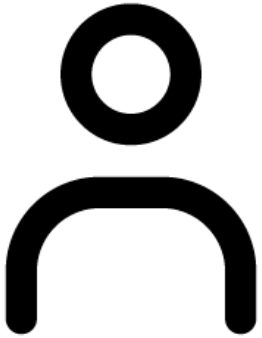
コマンドを扱う必要がある！

まずは復習してみましょう！

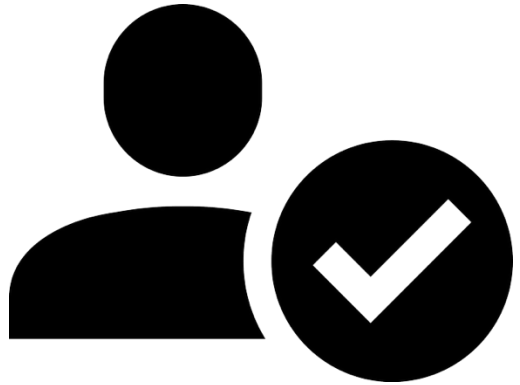
基本操作



復習



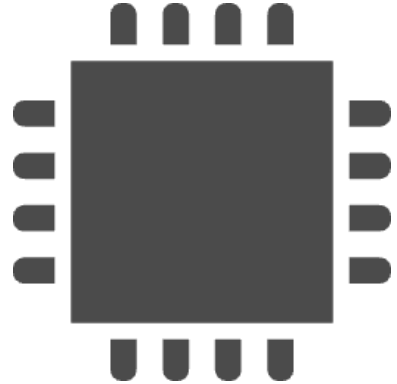
ユーザ・グループ



権限

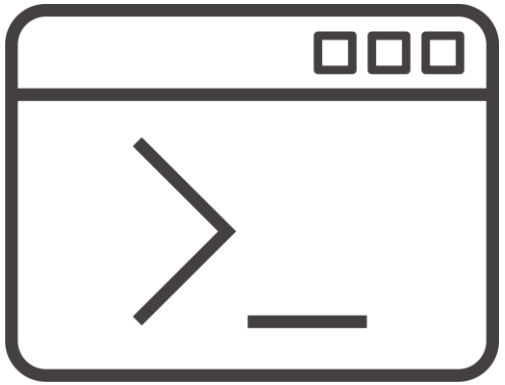
パッケージ

サービス



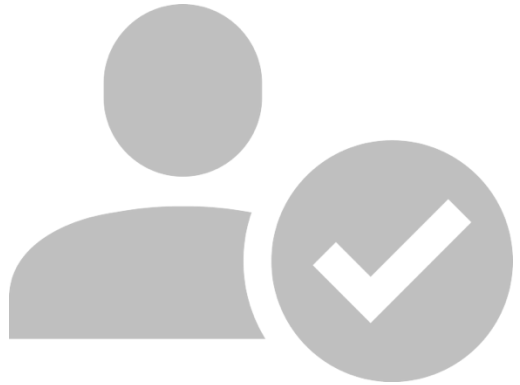
プロセス

基本操作



復習

ユーザ・グループ



権限

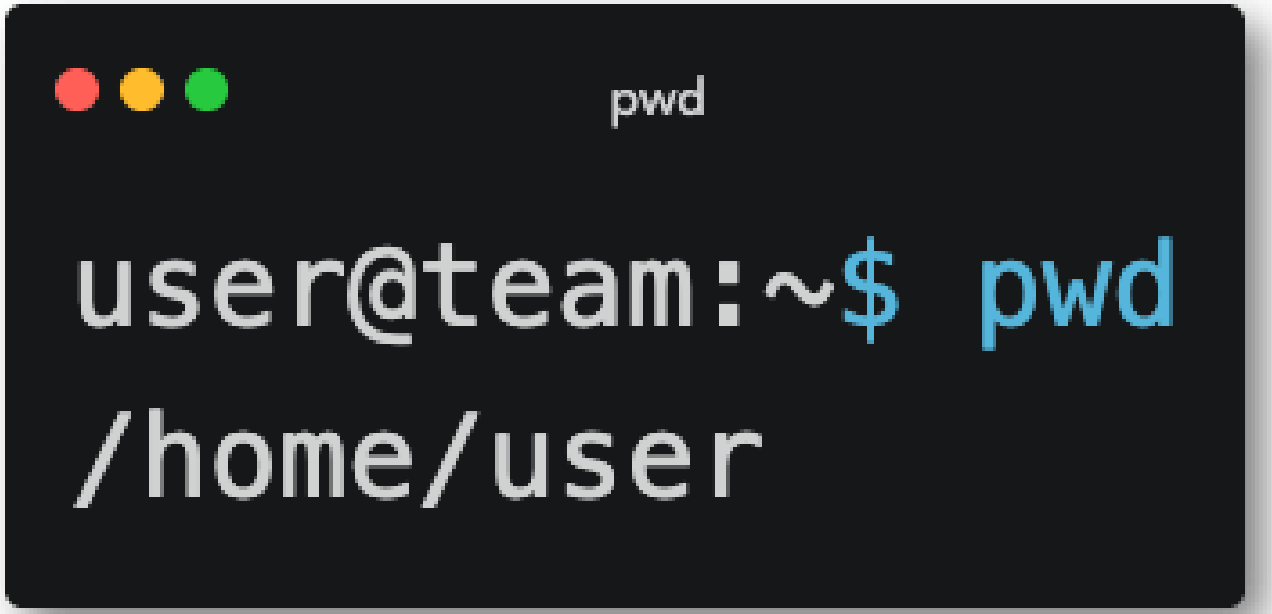
パッケージ

サービス

プロセス

基本的操作の復習：pwd

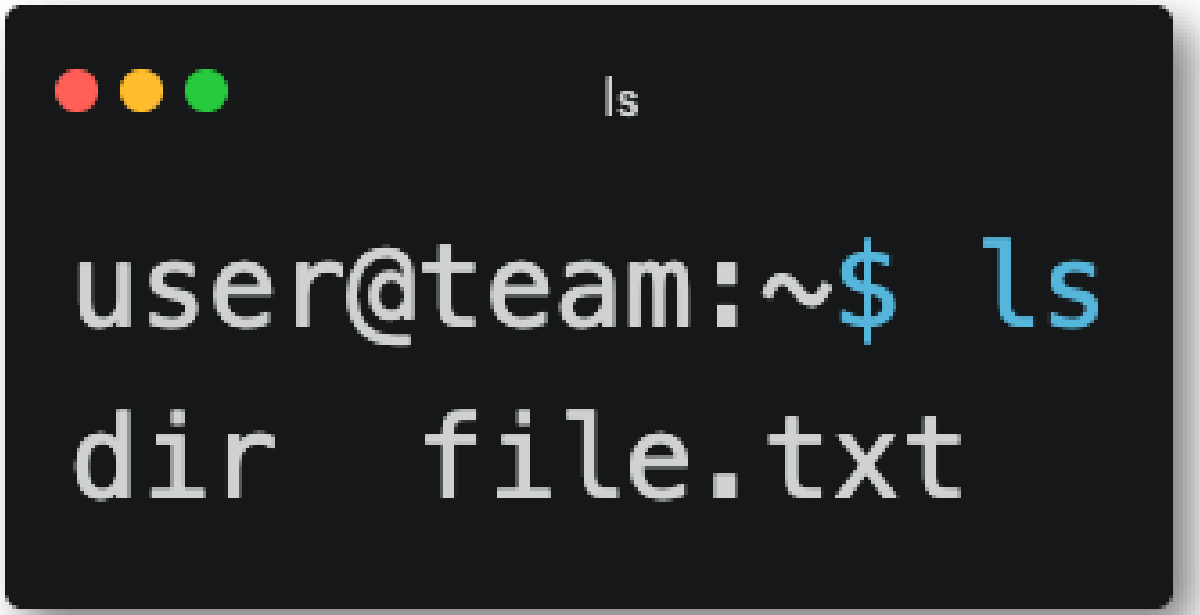
pwd (print working directory)



作業ディレクトリ（現在位置）を表示

基本的操作の復習：ls

ls (list)



ファイルやディレクトリの一覧を表示

基本的操作の復習：cd

cd (change directory)

カレントディレクトリ（現在位置）を変更

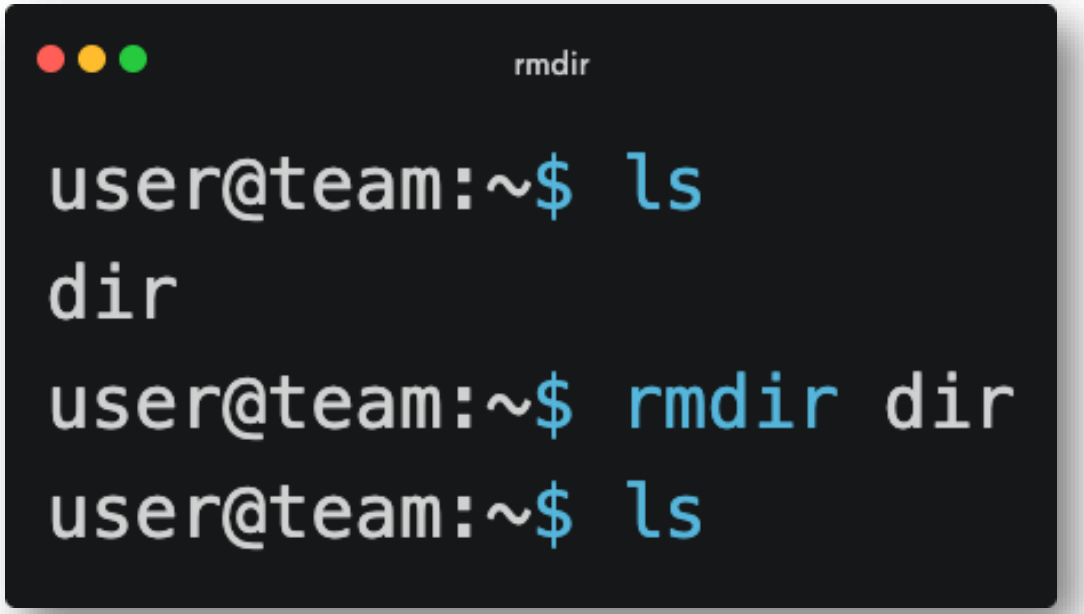
基本的操作の復習：mkdir

mkdir (make directory)

ディレクトリ（フォルダ）を作成

基本的操作の復習：rmdir

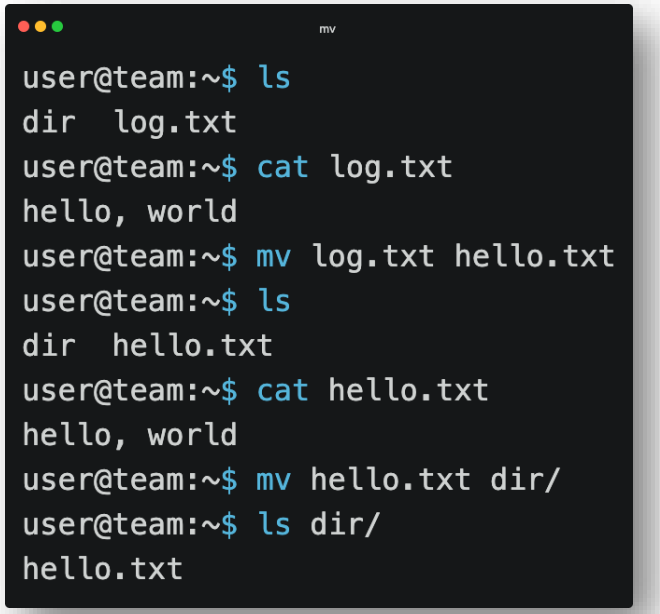
rmdir (remove directory)



空のディレクトリ（フォルダ）を削除

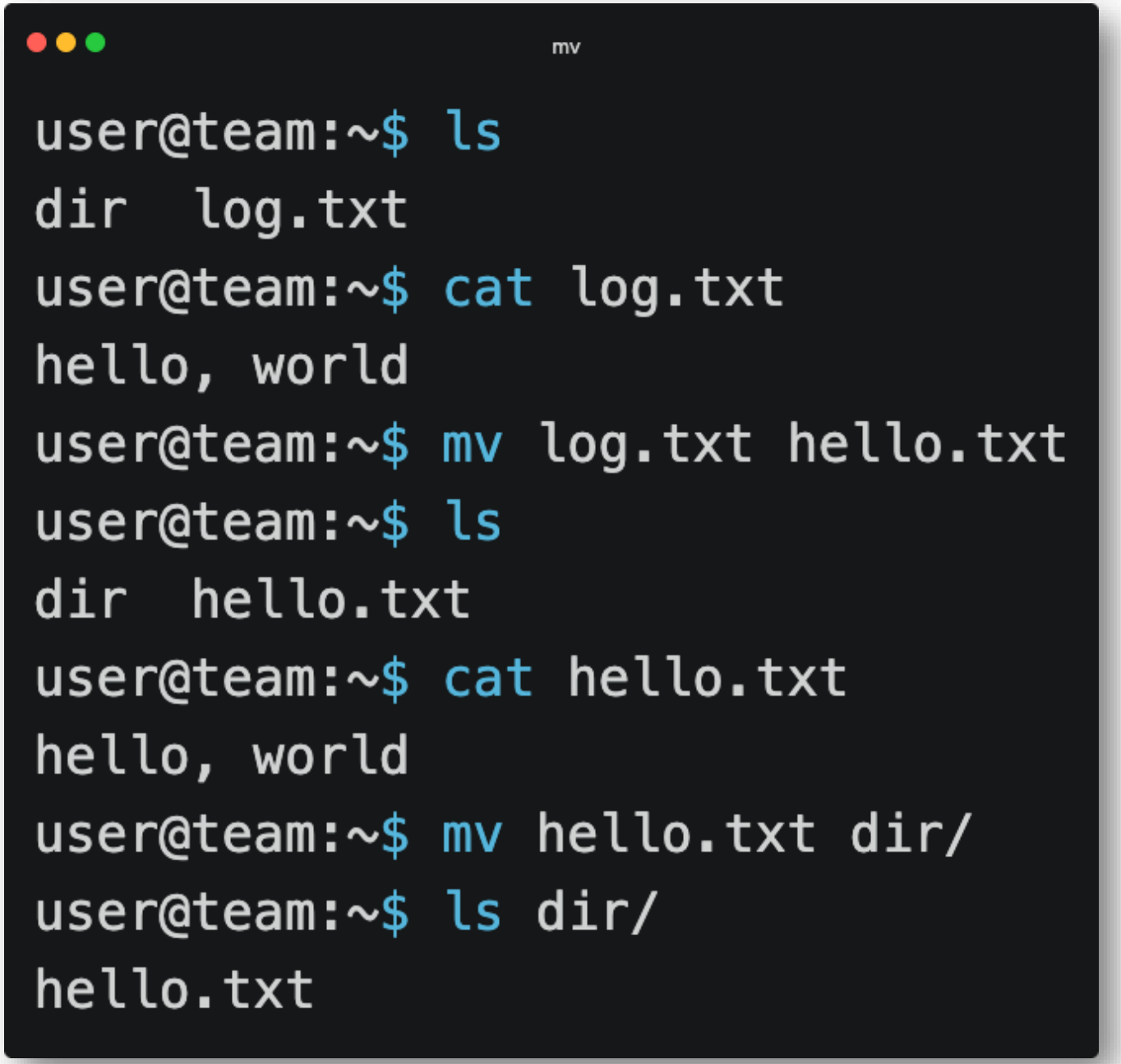
基本的操作の復習：mv

mv (move)



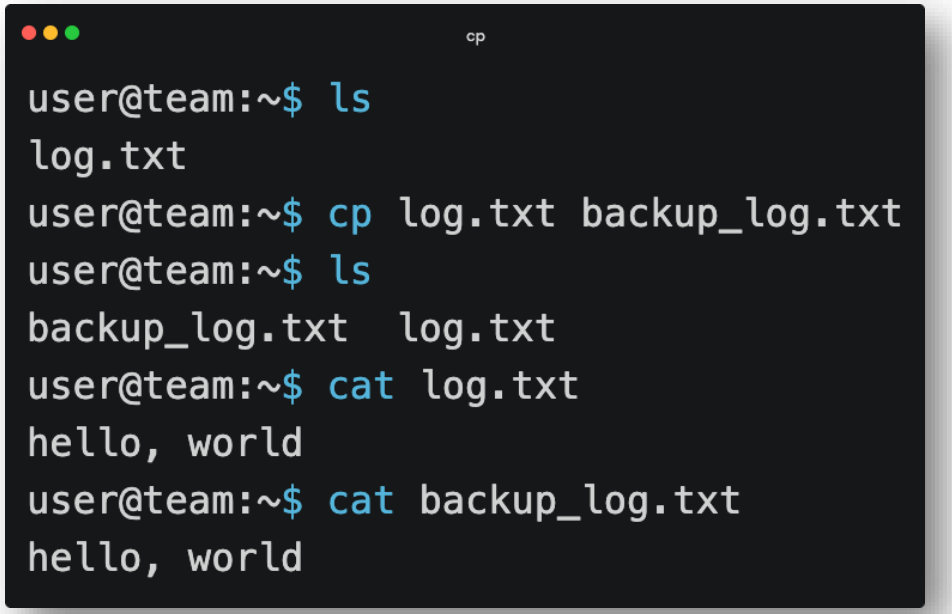
ファイルやディレクトリ（フォルダ）を移動・リネーム

基本的操作の復習：mv



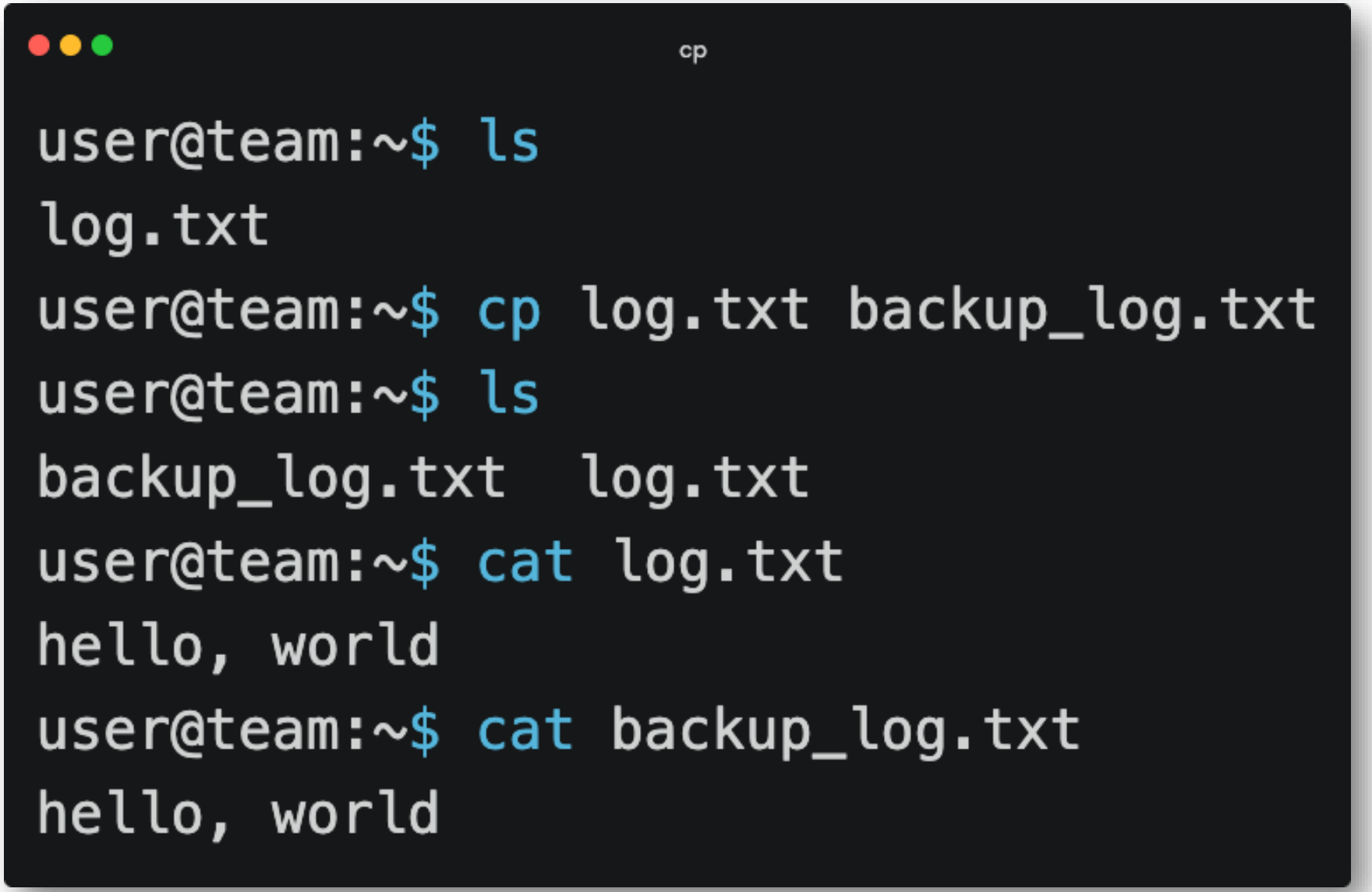
基本的操作の復習：cp

cp (copy)



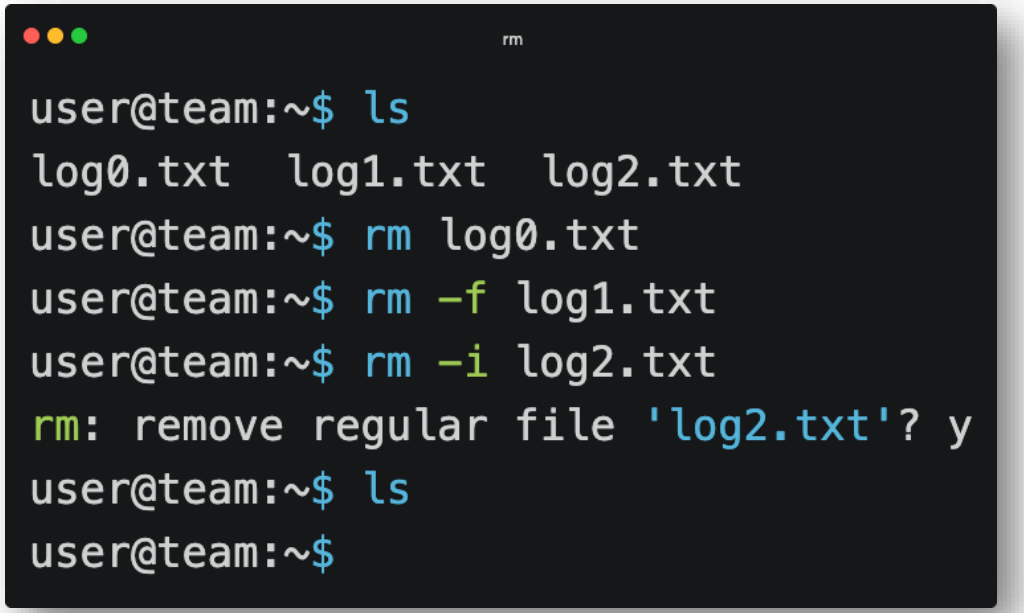
ファイルやディレクトリ（フォルダ）をコピーする

基本的操作の復習：cp



基本的操作の復習：rm

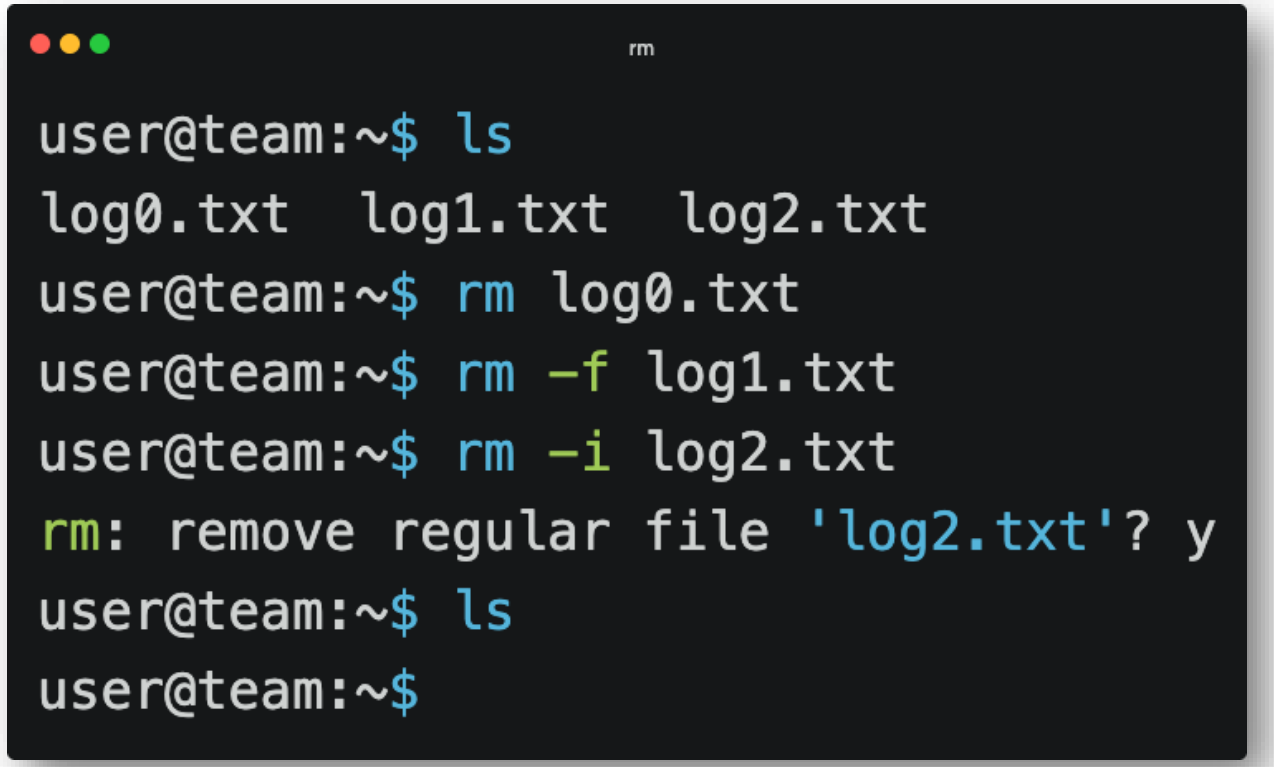
rm (remove)



ファイルやディレクトリ（フォルダ）を削除する

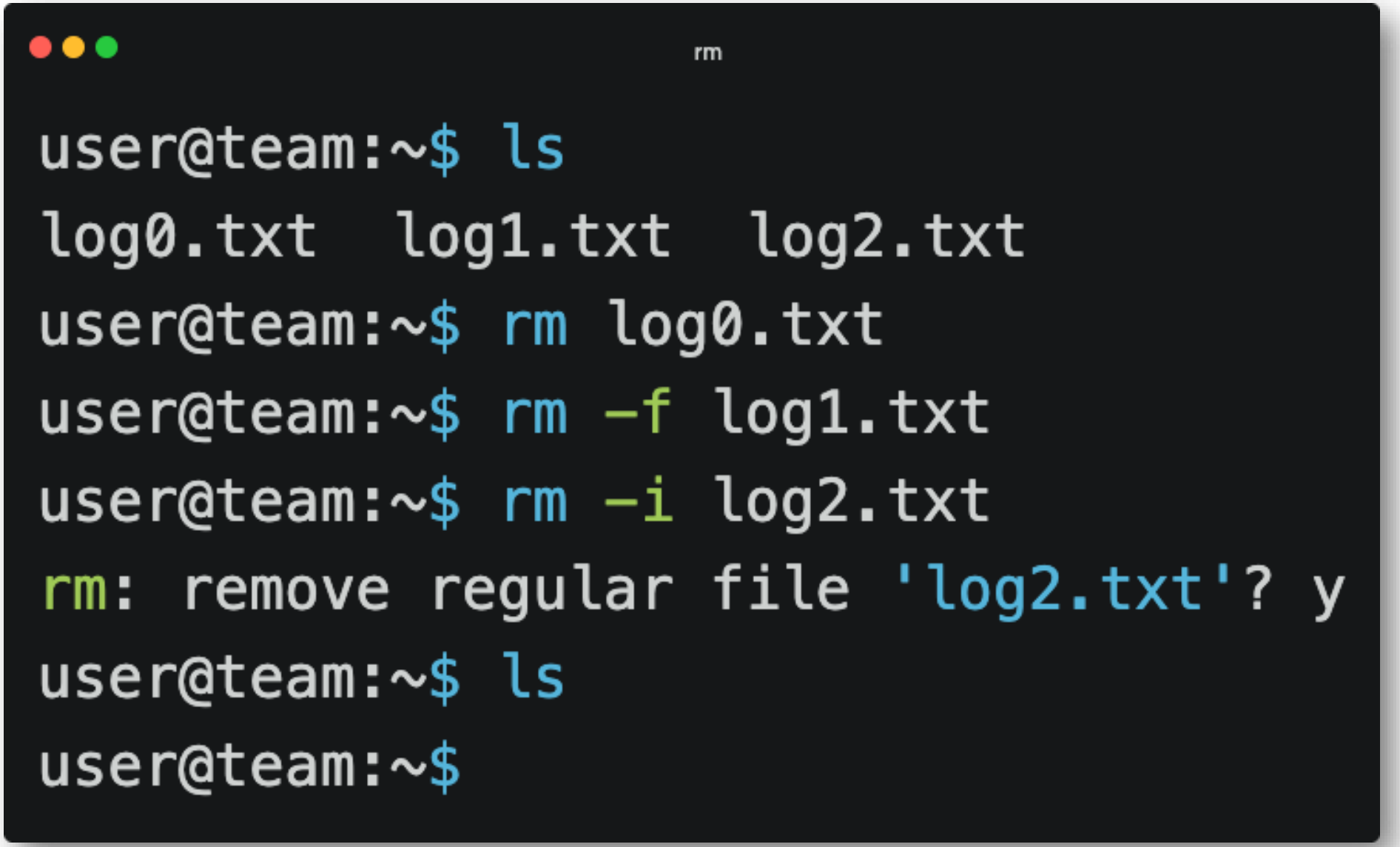
基本的操作の復習：rm

rm (remove)



オプション	意味
-f	強制削除
-i	確認

基本的操作の復習：rm



基本的操作の復習：演習

1. 現在いるディレクトリの絶対パス（完全なパス）を表示
2. 現在のディレクトリの内容を表示
3. 自分の名前のディレクトリ（例：hal）を作成
4. 作ったディレクトリに移動
5. touch test.txtでファイルを作成
6. ディレクトリのファイルを確認する
7. test.txtのファイル名を20250808.txtに変更
8. 20250808.txtをbackup.txtにコピー
9. 20250808.txtとbackup.txtを削除
10. 上のディレクトリに戻る
11. 自分のディレクトリを削除

基本的操作の復習：答え

Practice 0

```
user@host:~$ pwd
/home/user
user@host:~$ ls
user@host:~$ mkdir your_name
user@host:~$ cd your_name
user@host:~/your_name$ touch test.txt
user@host:~/your_name$ ls
test.txt
user@host:~/your_name$ mv test.txt 20250808.txt
user@host:~/your_name$ cp 20250808.txt backup.txt
user@host:~/your_name$ rm 20250808.txt
user@host:~/your_name$ rm backup.txt
user@host:~/your_name$ cd ..
user@host:~$ rmdir your_name
```

基本的操作の復習：答え

Practice 0

```
user@host:~$ pwd
/home/user
user@host:~$ ls
user@host:~$ mkdir your_name
user@host:~$ cd your_name
user@host:~/your_name$ touch test.txt
user@host:~/your_name$ ls
test.txt
user@host:~/your_name$ mv test.txt 20250808.txt
user@host:~/your_name$ cp 20250808.txt backup.txt
```

基本的操作の復習：答え

```
user@host:~$ cd your_name
user@host:~/your_name$ touch test.txt
user@host:~/your_name$ ls
test.txt
user@host:~/your_name$ mv test.txt 20250808.txt
user@host:~/your_name$ cp 20250808.txt backup.txt
user@host:~/your_name$ rm 20250808.txt
user@host:~/your_name$ rm backup.txt
user@host:~/your_name$ cd ..
user@host:~$ rmdir your_name
```

テキストエディタ

コマンド	名称	特徴
nano	GNU nano	• CtrlやAltとのショートカットキーを用いた操作 • ショートカットキーが常に表示されている
vi (vim)	Vim	• 入力モードや「:」を使ったコマンドによる操作 キーボードから手を離さずに操作が可能
emacs	Emacs	• CtrlやAltとのショートカットキーを用いた操作 • ショートカットキーは暗記
code	Visual Studio Code	• GUI環境のみ クライアントのGUIからSSH接続することが可能

お好みのものをお使いください

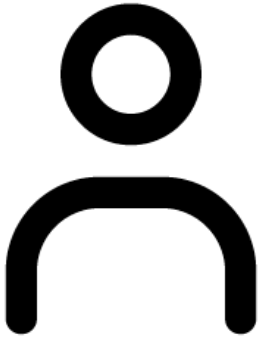
テキストエディタ

コマンド	名称	特徴
nano	GNU nano	• CtrlやAltとのショートカットキーを用いた操作 • ショートカットキーが常に表示されている
vi (vim)	Vim	• 入力モードや「:」を使ったコマンドによる操作 キーボードから手を離さずに操作が可能
emacs	Emacs	• CtrlやAltとのショートカットキーを用いた操作 • ショートカットキーは暗記
code	Visual Studio Code	• GUI環境のみ クライアントのGUIからSSH接続することが可能

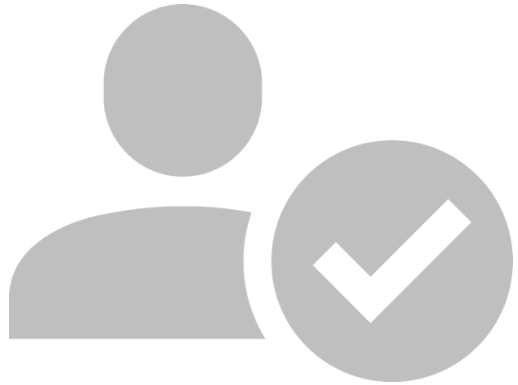
テキストの編集をしてみてください

基本操作

復習



ユーザ・グループ



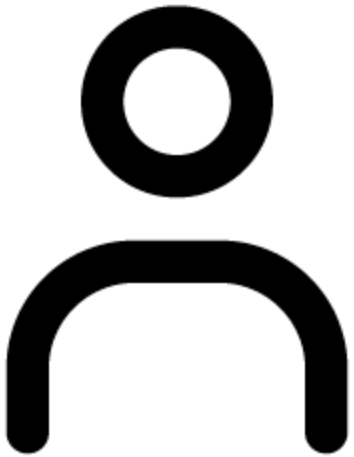
権限

パッケージ

サービス

プロセス

ユーザ



Linuxにログインして操作するためのアカウント

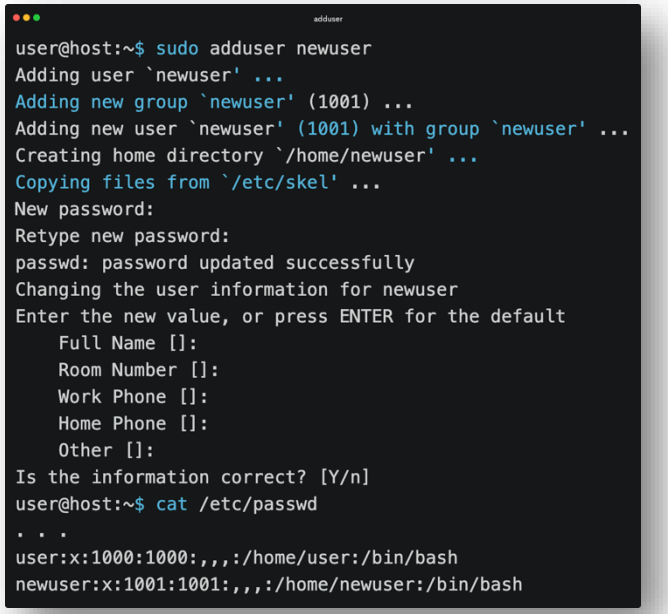
ユーザ作成：useradd

useradd

ユーザをコマンドだけで作成する
（一気に作るとき）

ユーザ作成：adduser

adduser



ユーザを対話しながら作成する
（一人ずつ作るとき）

ユーザ作成：adduser

adduser

```
user@host:~$ sudo adduser newuser
Adding user `newuser' ...
Adding new group `newuser' (1001) ...
Adding new user `newuser' (1001) with group `newuser' ...
Creating home directory `/home/newuser' ...
Copying files from `/etc/skel' ...
New password:
Retype new password:
passwd: password updated successfully
Changing the user information for newuser
Enter the new value, or press ENTER for the default
    Full Name []:
    Room Number []:
    Work Phone []:
    Home Phone []:
    Other []:
Is the information correct? [Y/n]
user@host:~$ cat /etc/passwd
. . .
user:x:1000:1000:,,,:/home/user:/bin/bash
newuser:x:1001:1001:,,,:/home/newuser:/bin/bash
```

ユーザ編集：usermod

usermod (user modify)

usermod

```
user@host:~$ sudo usermod --login newuser user
user@host:~$ sudo usermod --password newpassword user
user@host:~$ sudo usermod --home /home/newuser user
user@host:~$ sudo usermod --shell /bin/zsh user
user@host:~$ sudo usermod --expiredate 2025-08-09 user
```

ユーザを編集する

ユーザ編集：usermod

```
usermod
user@host:~$ sudo usermod --login newuser user
user@host:~$ sudo usermod --password newpassword user
user@host:~$ sudo usermod --home /home/newuser user
user@host:~$ sudo usermod --shell /bin/zsh user
user@host:~$ sudo usermod --expiredate 2025-08-09 user
```

オプション	意味
--login / -l	ユーザ名
--password / -p	パスワード
--home / -d	ホームディレクトリ
--shell / -s	シェル
--expiredate / -e	有効期限

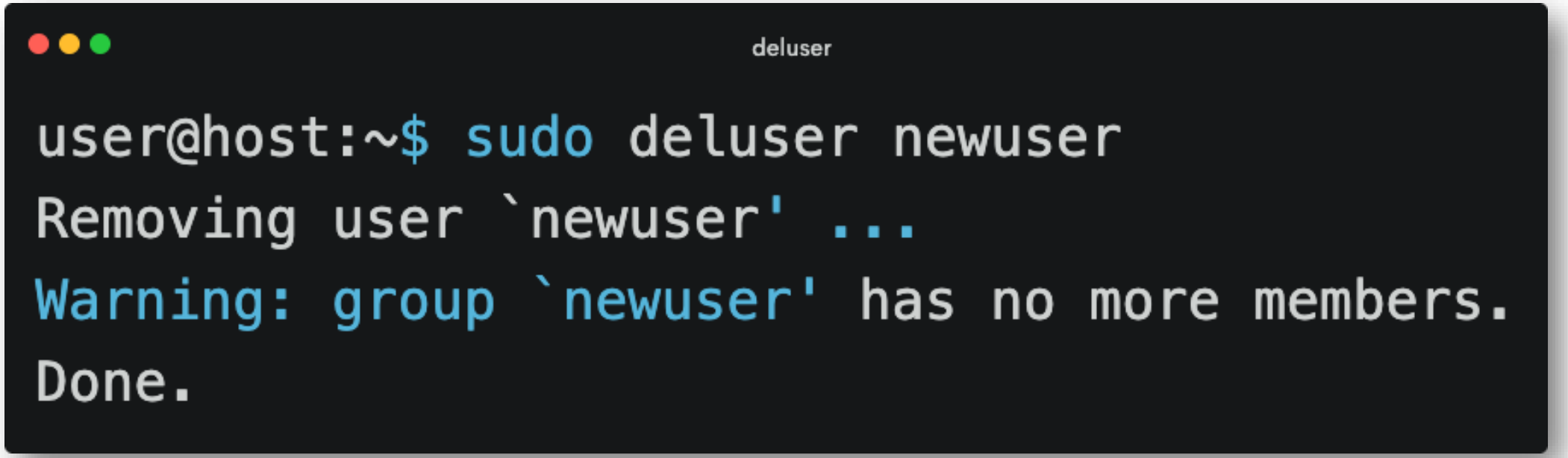
ユーザ削除：userdel

userdel (user delete)

ユーザをコマンドだけで削除する
（一気に消すとき）

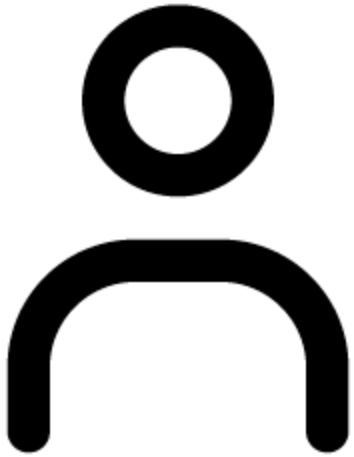
ユーザ削除：deluser

deluser (delete user)



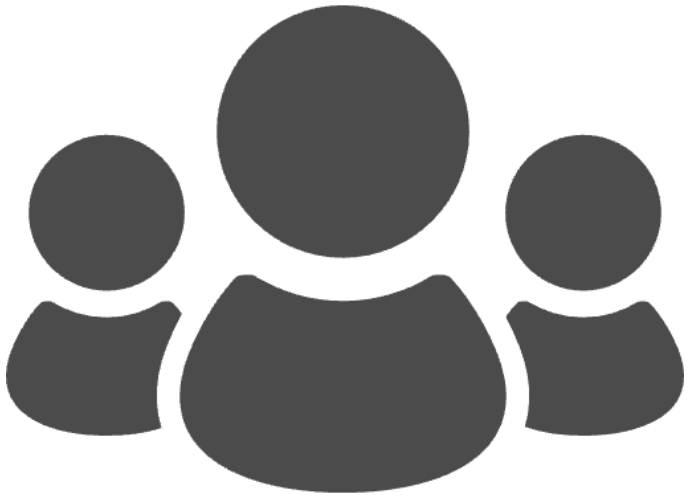
ユーザをオプションを指定して削除する
（一人ずつ消すとき）

ユーザ三種の神器



コマンド	内容
useradd	ユーザを作成する
usermod	ユーザを編集する
userdel	ユーザを削除する

グループとは



権限を一括で制御するために
複数のユーザをまとめた集団

すべてのユーザは
どれかのグループに所属している

- 複数のグループに所属できる
- 主グループ：1つのみ代表的なグループ
- 副グループ：いくつでも所属可能

グループ作成：groupadd

groupadd

グループをコマンドだけで作成する
（一気に作るとき）

グループ作成：addgroup

addgroup

グループを対話しながら作成する
（1Gずつ作るとき）

グループ編集：groupmod

groupmod (group modify)

groupmod

```
user@host:~$ sudo groupmod -n newgroupname oldgroupname
user@host:~$ cat /etc/group
root:x:0:
. . .
newgroupname:x:1001:
```

グループを編集する
--new-name / -n: 新しいグループ名

グループ削除：groupdel

groupdel (group delete)

グループを削除する

ユーザをグループに追加する：usermod -aG *group user*

usermod –aG *group user*

usermod-group

```
user@host:~$ sudo usermod -aG newgroup user
user@host:~$ groups user
user : user newgroup
```

*user*を*group*に追加で所属させる

ユーザ操作・グループ操作のまとめ

コマンド	内容
useradd / adduser	ユーザを追加する
usermod	ユーザを編集する
userdel / deluser	ユーザを削除する
groupadd / addgroup	グループを追加する
groupmod	グループを編集する
groupdel	グループを削除する
usermod –aG *group user*	*user*を*group*に追加で所属させる

ユーザ・グループ：演習

1. adduserで自分の名前（例：hal）のユーザを作る
2. ユーザが作成されたか確認する
3. 好きなフルーツ（例：apple）のグループを作る
4. 先ほどのユーザをそのグループに追加する
5. 先ほどのユーザの所属グループを確認する

ユーザ・グループ：演習

User Group Practice

```
user@host:~$ sudo adduser your_name
Adding user `your_name' ...
Adding new group `your_name' (1001) ...
Adding new user `your_name' (1001) with group `your_name'
...
Creating home directory `/home/your_name' ...
Copying files from `/etc/skel' ...
New password: aic0808
Retype new password: aic0808
passwd: password updated successfully
Changing the user information for your_name
Enter the new value, or press ENTER for the default
    Full Name []:
    Room Number []:
    Work Phone []:
    Home Phone []:
    Other []:
Is the information correct? [Y/n]
user@host:~$ cat /etc/passwd
. . .
user:x:1000:1000:,,,:/home/user:/bin/bash
your_name:x:1001:1001:,,,:/home/your_name:/bin/bash
user@host:~$ sudo addgroup your_fruit
Adding group `your_fruit' (GID 1002) ...
Done.
user@host:~$ cat /etc/group
root:x:0:
. . .
your_fruit:x:1002:
user@host:~$ sudo usermod -aG your_fruit your_name
user@host:~$ groups your_name
your_name : your_name your_fruit
```

ユーザ・グループ：演習

User Group Practice

```
user@host:~$ sudo adduser your_name
Adding user `your_name' ...
Adding new group `your_name' (1001) ...
Adding new user `your_name' (1001) with group `your_name'
...
Creating home directory `/home/your_name' ...
Copying files from `/etc/skel' ...
New password: aic0808
Retype new password: aic0808
passwd: password updated successfully
Changing the user information for your_name
Enter the new value, or press ENTER for the default
    Full Name []:
    Room Number []:
```

ユーザ・グループ：演習

```
passwd: password updated successfully
Changing the user information for your_name
Enter the new value, or press ENTER for the default
    Full Name []:
    Room Number []:
    Work Phone []:
    Home Phone []:
    Other []:
Is the information correct? [Y/n]
user@host:~$ cat /etc/passwd
. . .
user:x:1000:1000:,,,:/home/user:/bin/bash
your_name:x:1001:1001:,,,:/home/your_name:/bin/bash
user@host:~$ sudo addgroup your_fruit
Adding group `your_fruit' (GID 1002) ...
```

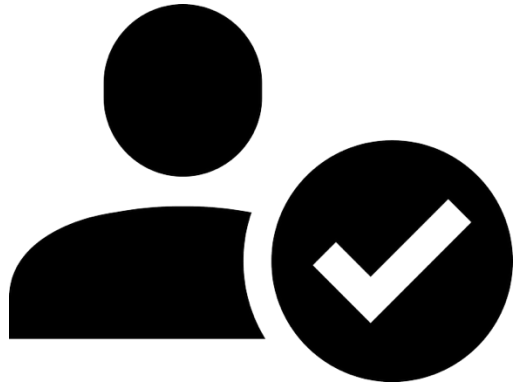
ユーザ・グループ：演習

```
. . .
user:x:1000:1000:,,,:/home/user:/bin/bash
your_name:x:1001:1001:,,,:/home/your_name:/bin/bash
user@host:~$ sudo addgroup your_fruit
Adding group `your_fruit' (GID 1002) ...
Done.
user@host:~$ cat /etc/group
root:x:0:
. . .
your_fruit:x:1002:
user@host:~$ sudo usermod -aG your_fruit your_name
user@host:~$ groups your_name
your_name : your_name your_fruit
```

基本操作

復習

ユーザ・グループ



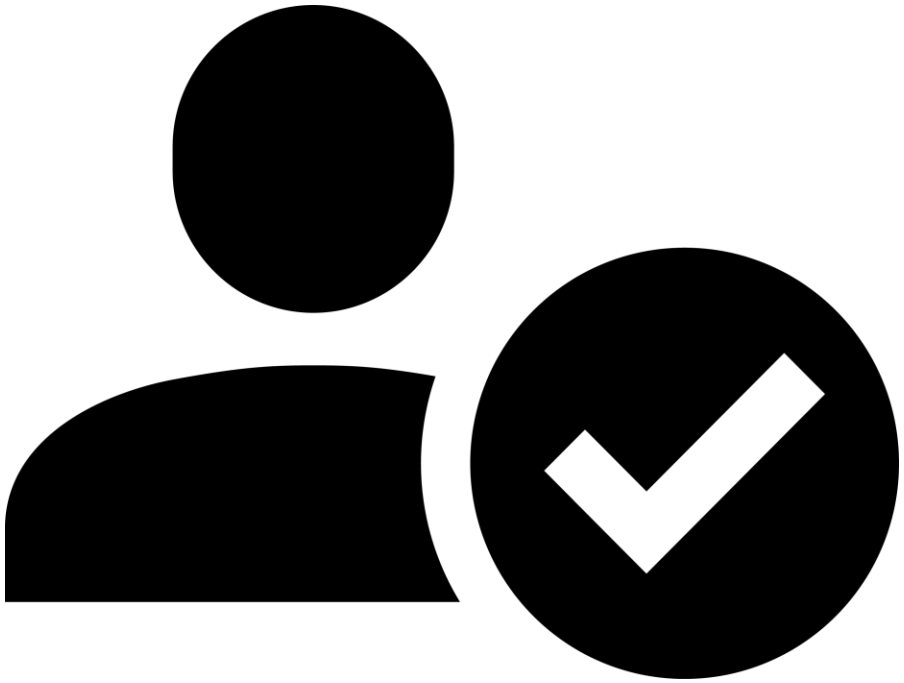
権限

パッケージ

サービス

プロセス

パーミッション

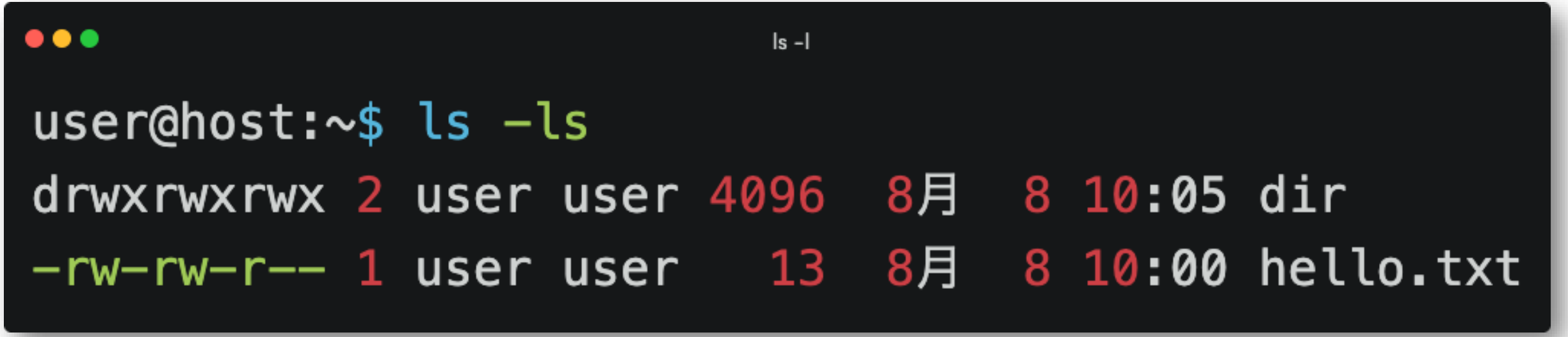


ファイルやディレクトリを
誰が持っていて

誰が
- 読み取り (Read)
- 書き込み (Write)
- 実行 (eXecute)

可能か

パーミッションの読み方



ファイルやディレクトリには
所有ユーザと所有グループがある

パーミッションの読み方

```
drwxrwxrwx 2 user user 4
-rw-rw-r-- 1 user user
```

文字	d	rwx	rwx	rwx
意味	種類	所有ユーザ（User）	所有グループ（Group）	その他（Other）
内容	• -: ファイル • d: ディレクトリ • l: リンク	• r: 読み込み可能 • w: 書き込み可能 • x: 実行可能	• r: 読み込み可能 • w: 書き込み可能 • x: 実行可能	• r: 読み込み可能 • w: 書き込み可能 • x: 実行可能

オーナー変更：chown

chown (change owner)

chown

```
user@host:~$ ls -l
drwxrwxrwx 2 user user 4096  8月  8 10:05 dir
-rw-rw-r-- 1 user user   13  8月  8 10:00 hello.txt
user@host:~$ sudo chown root hello.txt
user@host:~$ ls -l
drwxrwxrwx 2 user user 4096  8月  8 10:05 dir
-rw-r--r-- 1 root user   13  8月  8 10:00 hello.txt
user@host:~$ sudo chown root:root hello.txt
user@host:~$ ls -l
drwxrwxrwx 2 user user 4096  8月  8 10:05 dir
-rw-r--r-- 1 root root   13  8月  8 10:00 hello.txt
```

所有者を変更

オーナー変更：chown

所有者を変更

```
chown
user@host:~$ ls -l
drwxrwxrwx 2 user user 4096  8月  8 10:05 dir
-rw-rw-r-- 1 user user   13  8月  8 10:00 hello.txt
user@host:~$ sudo chown root hello.txt
user@host:~$ ls -l
drwxrwxrwx 2 user user 4096  8月  8 10:05 dir
-rw-r--r-- 1 root user   13  8月  8 10:00 hello.txt
user@host:~$ sudo chown root:root hello.txt
user@host:~$ ls -l
drwxrwxrwx 2 user user 4096  8月  8 10:05 dir
-rw-r--r-- 1 root root   13  8月  8 10:00 hello.txt
```

- chown *user file...*

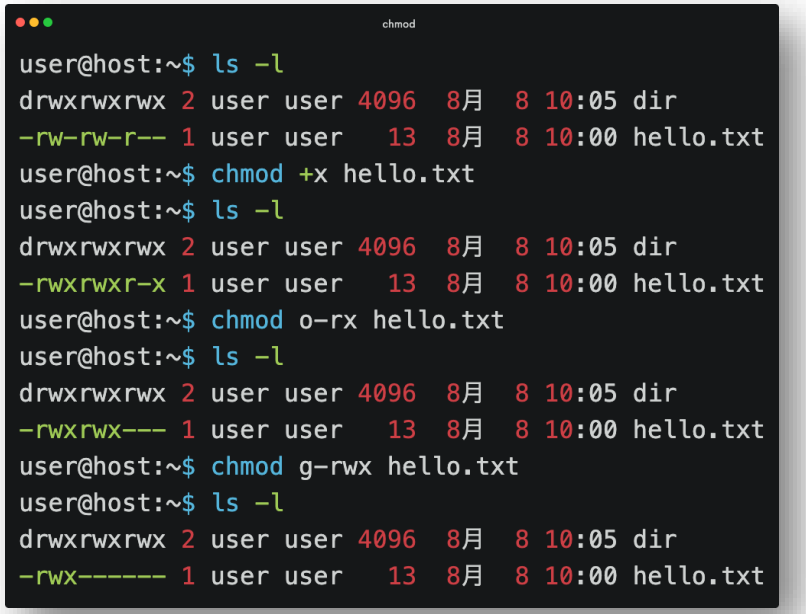
  所有ユーザを変更

- chown *user*:*group file...*

  所有ユーザもグループも変更

パーミッション変更：chmod

chmod (change mode)



権限を変更

権限を変更

```
chmod
user@host:~$ ls -l
drwxrwxrwx 2 user user 4096  8月  8 10:05 dir
-rw-rw-r-- 1 user user   13  8月  8 10:00 hello.txt
user@host:~$ chmod +x hello.txt
user@host:~$ ls -l
drwxrwxrwx 2 user user 4096  8月  8 10:05 dir
-rwxrwxr-x 1 user user   13  8月  8 10:00 hello.txt
user@host:~$ chmod o-rx hello.txt
user@host:~$ ls -l
drwxrwxrwx 2 user user 4096  8月  8 10:05 dir
-rwxrwx--- 1 user user   13  8月  8 10:00 hello.txt
user@host:~$ chmod g-rwx hello.txt
user@host:~$ ls -l
drwxrwxrwx 2 user user 4096  8月  8 10:05 dir
-rwx------ 1 user user   13  8月  8 10:00 hello.txt
```

- chmod ugoa+-rwx file...

  権限を変更

- 対象（ugoa）に

  権限（rwx）を

  与えるか除くか

管理者権限

常になんでもできると危ない

管理者権限

常になんでもできると危ない

そこで必要なときだけ特権を
使えるようにする

管理者として実行：sudo

sudo (super user do)

管理者としてコマンドを実行する

sudo：実験

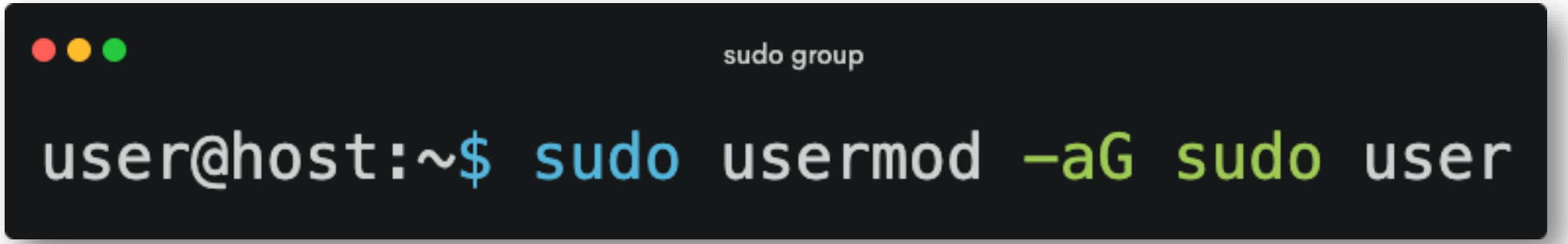
先ほど作ったユーザでログインしてsudoしてみる

sudo：実験

sudoはsudoグループに入ってるユーザが使える

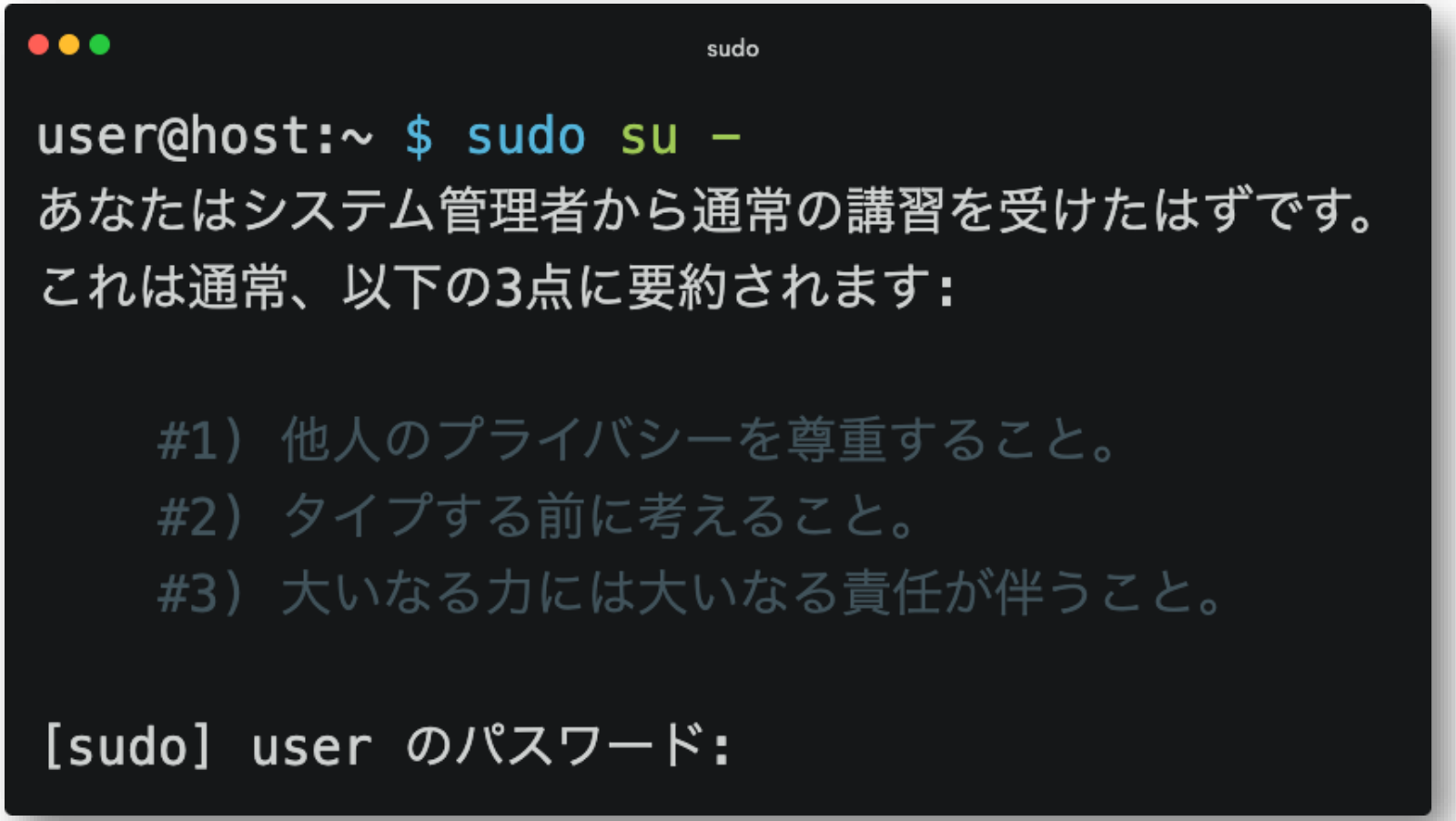
管理者を指定：sudo usermod -aG sudo *user*

usermod –aG sudo *user*



*user*をsudoグループに追加する

sudoした時の扱い



特権を持っているrootユーザとして実行することになる

パーミッションまとめ

コマンド	内容
chown	所有者の変更
chmod	権限の変更
sudo *command*	管理者として実行
sudo usermod –aG sudo *user*	*user*を管理者グループに追加

パーミッション：演習

1. 自分のユーザがsudoを使えるようにする
2. 自分のユーザでログインし直す
3. touch perm.txtでファイルを作る
4. 全員閲覧可能で、自分だけ編集可能なファイルにする
5. 権限を確認する
6. perm.txtの所有者をroot:rootにする

パーミッション：演習

Permission Practice

```
user@host:~$ sudo usermod -aG sudo your_name
user@host:~$ exit
user@local:~$ ssh your_name@host
Last login: Mon Aug  8 10:00:00 2025 from 131.113.x.x
your_name@host:~$ touch perm.txt
your_name@host:~$ chmod a+r perm.txt
your_name@host:~$ chmod go-w perm.txt
your_name@host:~$ chmod 644 perm.txt
your_name@host:~$ ls -l perm.txt
-rw-r--r-- 1 your_name your_name 0 Aug  8 10:00 perm.txt
your_name@host:~$ sudo chown root:root perm.txt
your_name@host:~$ ls -l perm.txt
-rw-r--r-- 1 root root 0 Aug  8 10:00 perm.txt
```

docker：準備

dockerを使えるか確認する

$ docker ps

権限不足で弾かれる

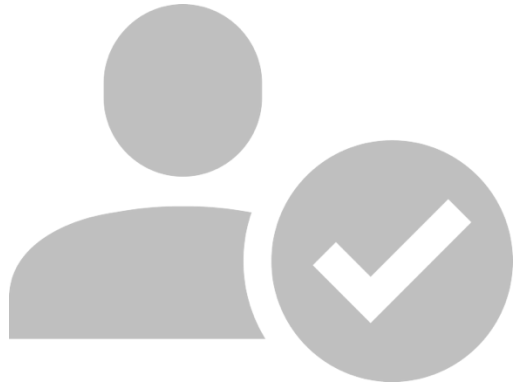
```
docker ps
your_name@host:~$ docker ps
permission denied while trying to connect to the Docker
daemon socket at unix:///var/run/docker.sock: Get
"http:///var/run/docker.sock/v1.51/containers/json": dial
unix /var/run/docker.sock: connect: permission denied

your_name@host:~$ sudo usermod -aG docker your_name
your_name@host:~$ groups your_name
your_name : your_name docker
```

$ sudo usermod –aG docker your_name
権限を得よう！

基本操作

復習

ユーザ・グループ



権限

パッケージ

サービス

プロセス

パッケージとは

大雑把にはプログラムのこと。

詳しくいうと、
機械語が同じハードウェアなら
同じバイナリ（機械語）が使えるから、
共有するためのまとまり。

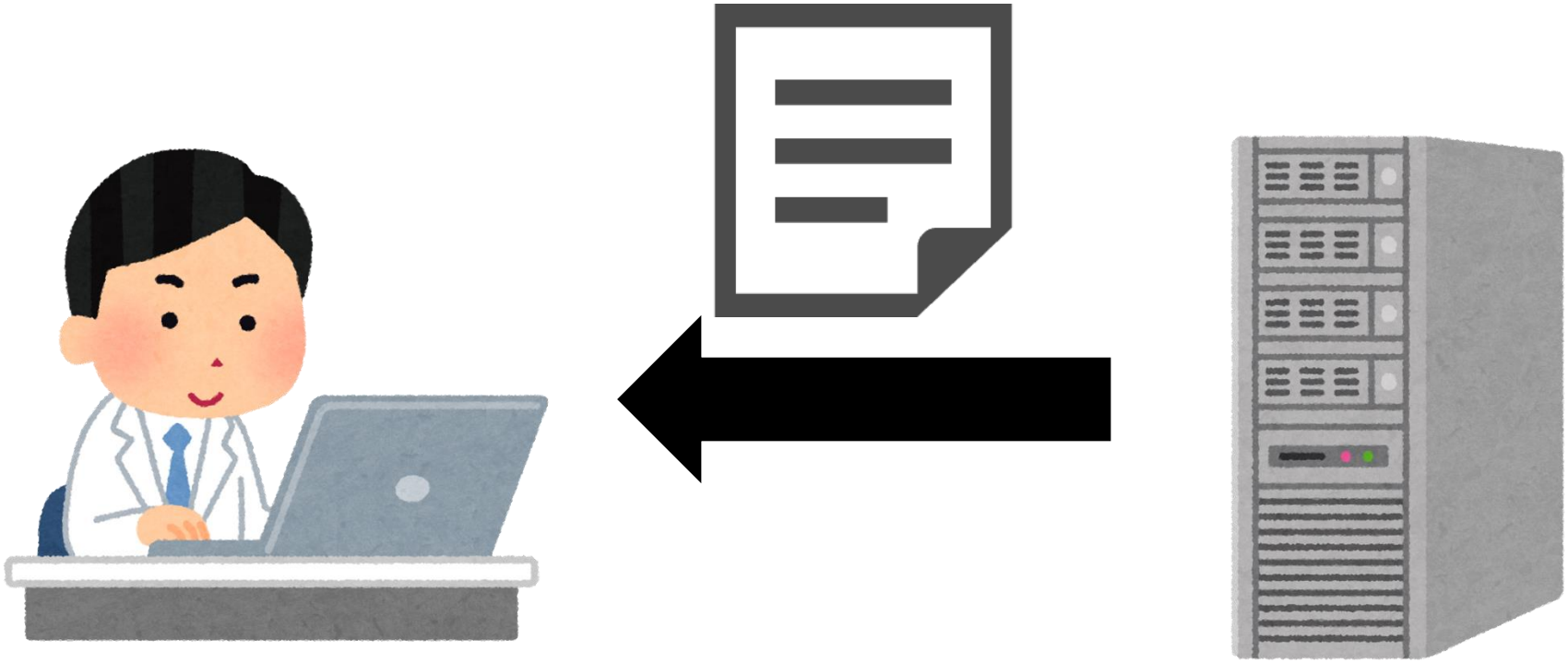
プログラムのインストールに使う。

Debian系のパッケージマネージャー：apt

Ubuntuを含むDebian系OSは、
aptというパッケージマネージャを使うことが多い。

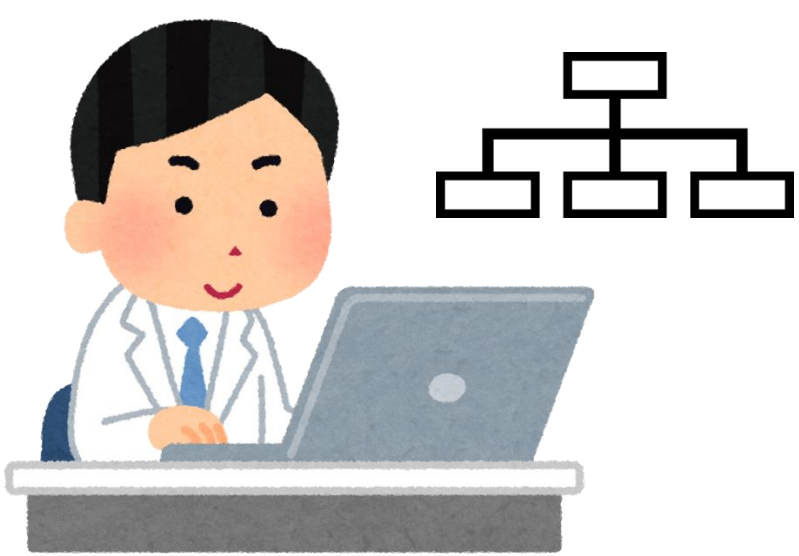
https://www.reddit.com/r/linux/comments/1czajn0/linux_distro_family_chart_with_distros_based/

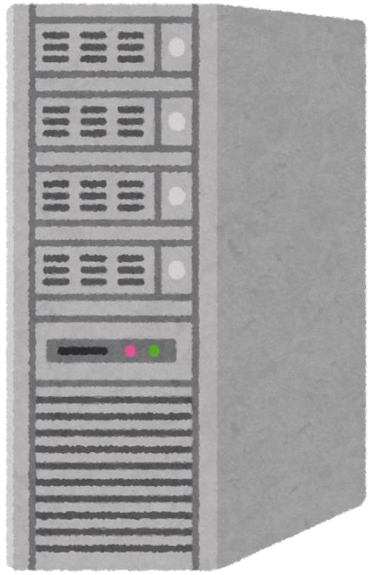
aptの仕組み



レポジトリから一覧を取得

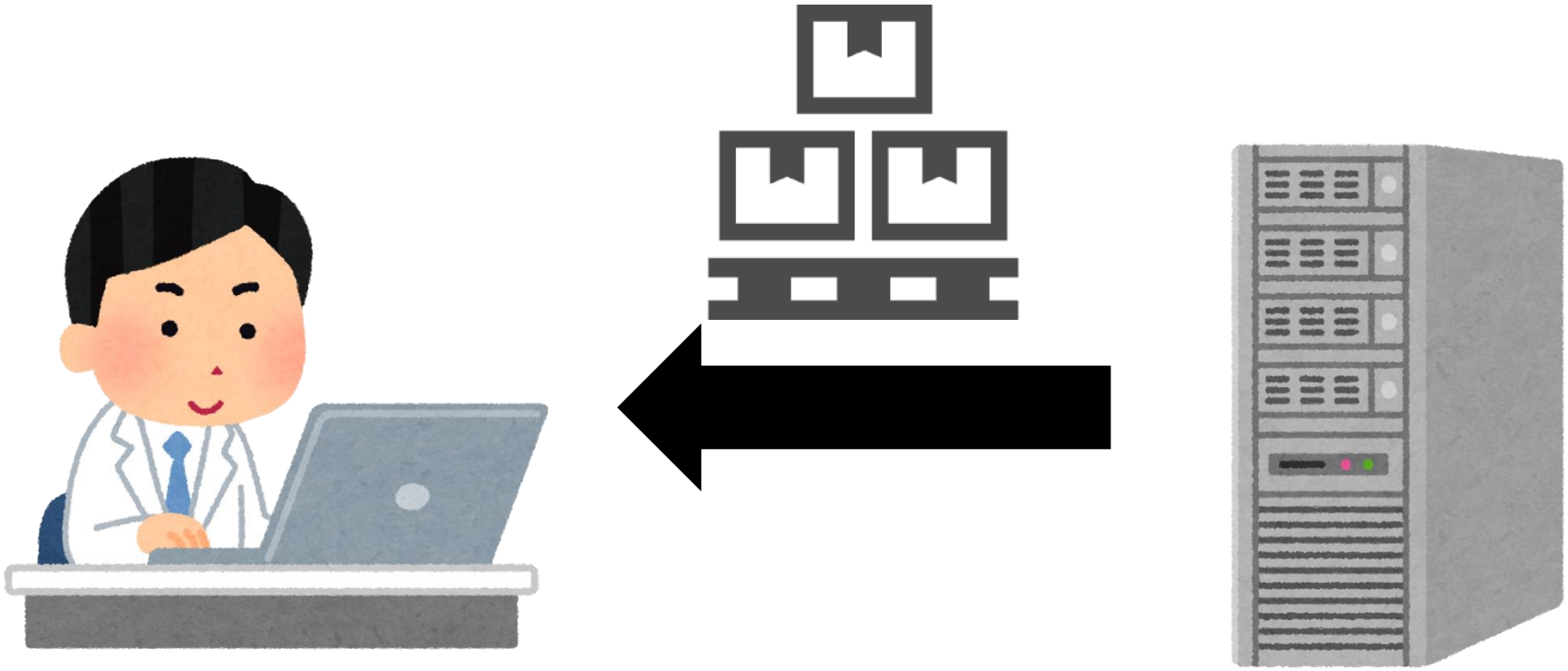
aptの仕組み





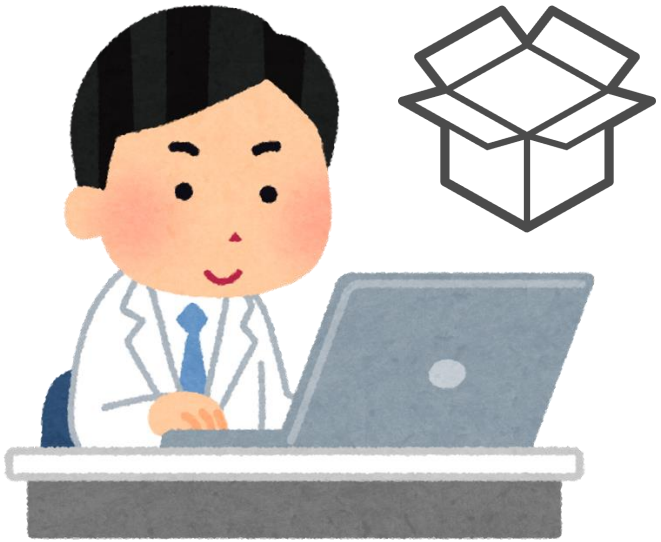
依存しているパッケージをすべて考える

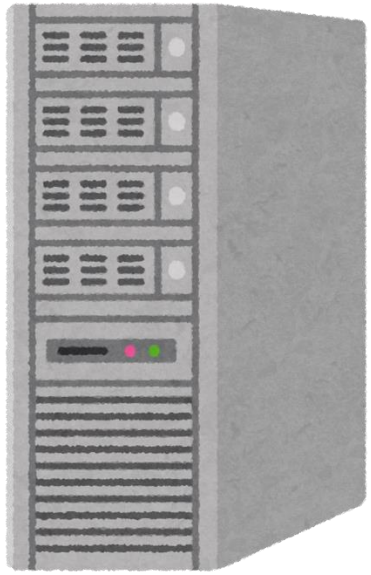
aptの仕組み



レポジトリからパッケージの圧縮ファイルをダウンロード

aptの仕組み

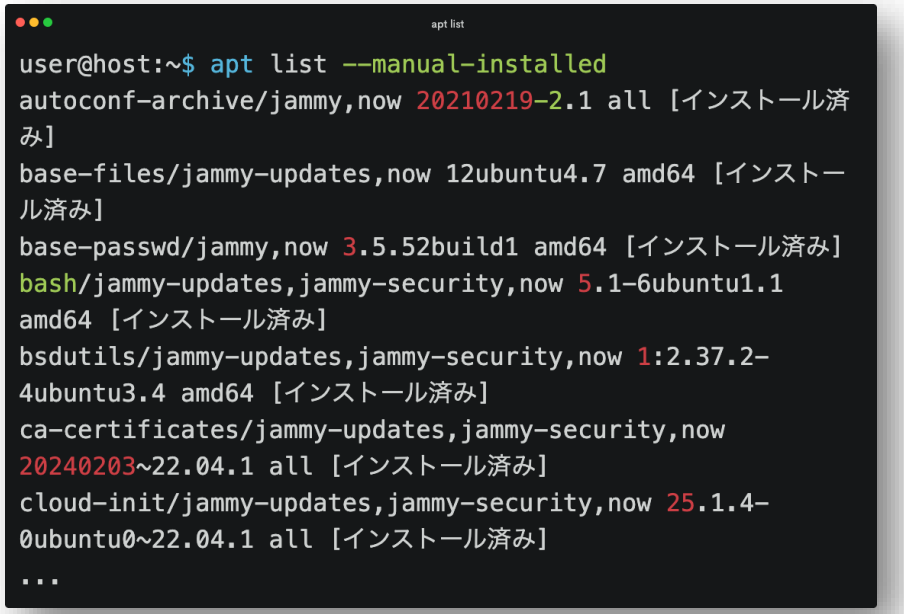




ダウンロードした圧縮ファイルを展開

パッケージ一覧の確認：apt list

apt list [pkg]



aptが把握している全てのパッケージ一覧を表示する

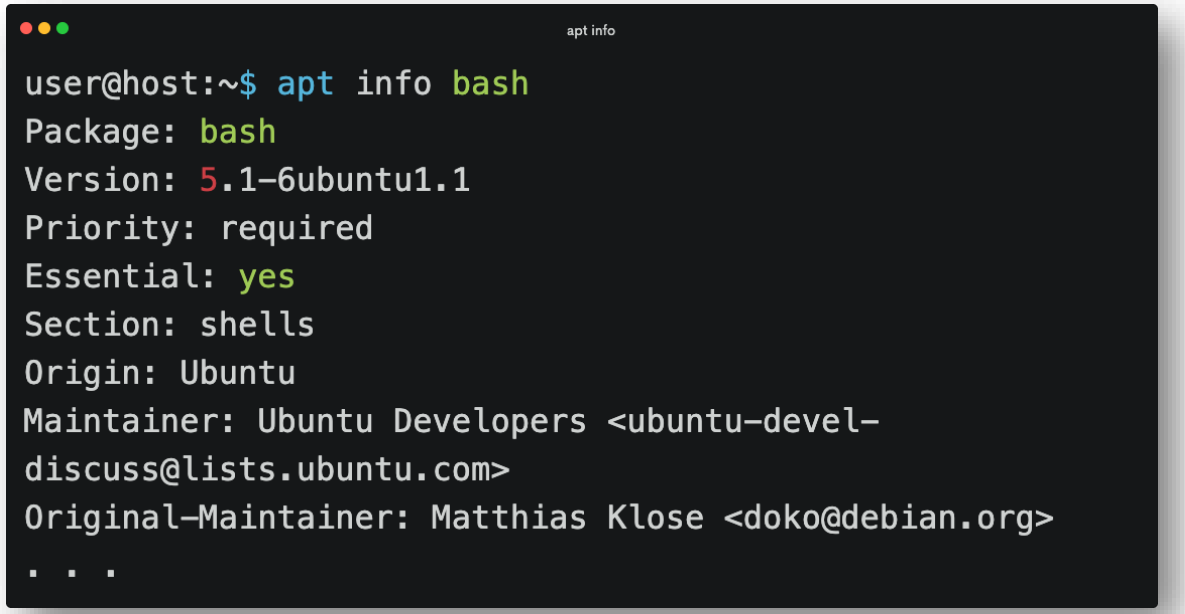
パッケージ一覧の確認：apt list

```
apt list
user@host:~$ apt list --manual-installed
autoconf-archive/jammy,now 20210219-2.1 all [インストール済み]
base-files/jammy-updates,now 12ubuntu4.7 amd64 [インストール済み]
base-passwd/jammy,now 3.5.52build1 amd64 [インストール済み]
bash/jammy-updates,jammy-security,now 5.1-6ubuntu1.1 amd64 [インストール済み]
bsdutils/jammy-updates,jammy-security,now 1:2.37.2-4ubuntu3.4 amd64 [インストール済み]
ca-certificates/jammy-updates,jammy-security,now 20240203~22.04.1 all [インストール済み]
cloud-init/jammy-updates,jammy-security,now 25.1.4-0ubuntu0~22.04.1 all [インストール済み]
...
```

オプション	意味
[pkg]	名前が一致するもの
--installed	インストールされているもの
--manual-installed	自分がインストールを指定したもの
--upgradable	アップグレード可能なもの

パッケージ情報の確認：apt info *pkg*

apt info *pkg*



*pkg*についての情報を表示する

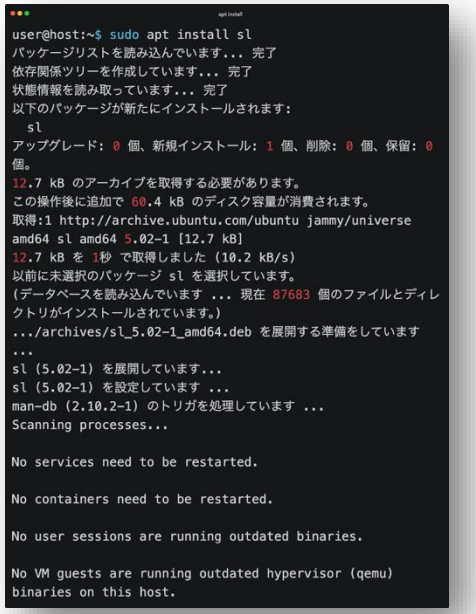
パッケージリストの更新：apt update

apt update

aptのパッケージリストを更新する

パッケージのインストール：apt install *pkg*

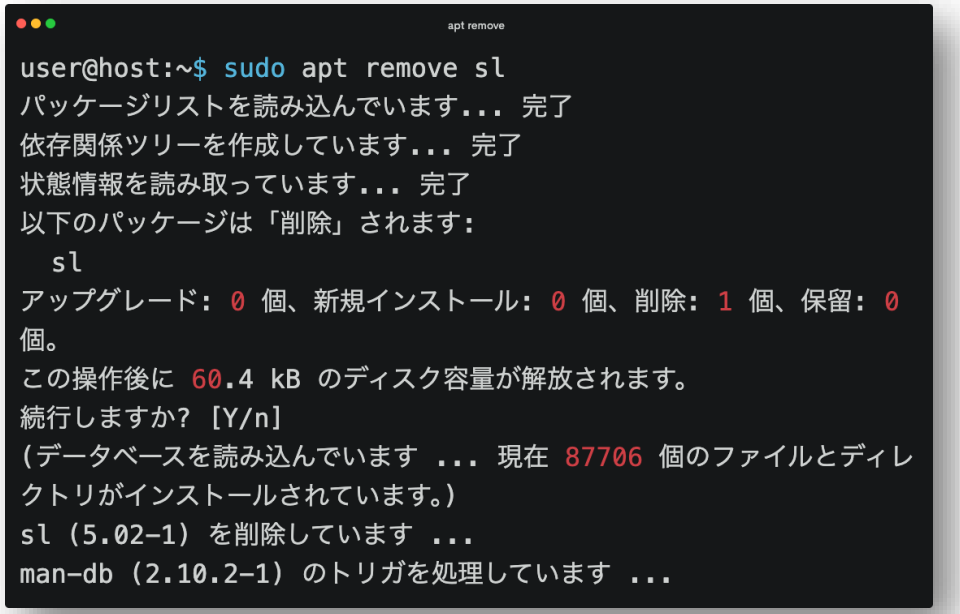
apt install *pkg*



*pkg*をインストールする

パッケージの削除：apt remove *pkg*

apt remove *pkg*



*pkg*をアンインストールする

パッケージの自動削除：apt autoremove

apt autoremove

使われていない依存パッケージをアンインストールする

パッケージの一括更新：apt upgrade

apt upgrade

すべてのパッケージを更新する

aptまとめ

コマンド	内容
apt list	パッケージ一覧を表示
apt info *pkg*	*pkg*の情報を表示
apt update	パッケージリストを更新
apt install *pkg*	*pkg*をインストール
apt remove *pkg*	*pkg*をアンインストール
apt upgrade	すべてのパッケージをアップデート

各チームで1人ずつ実行しましょう

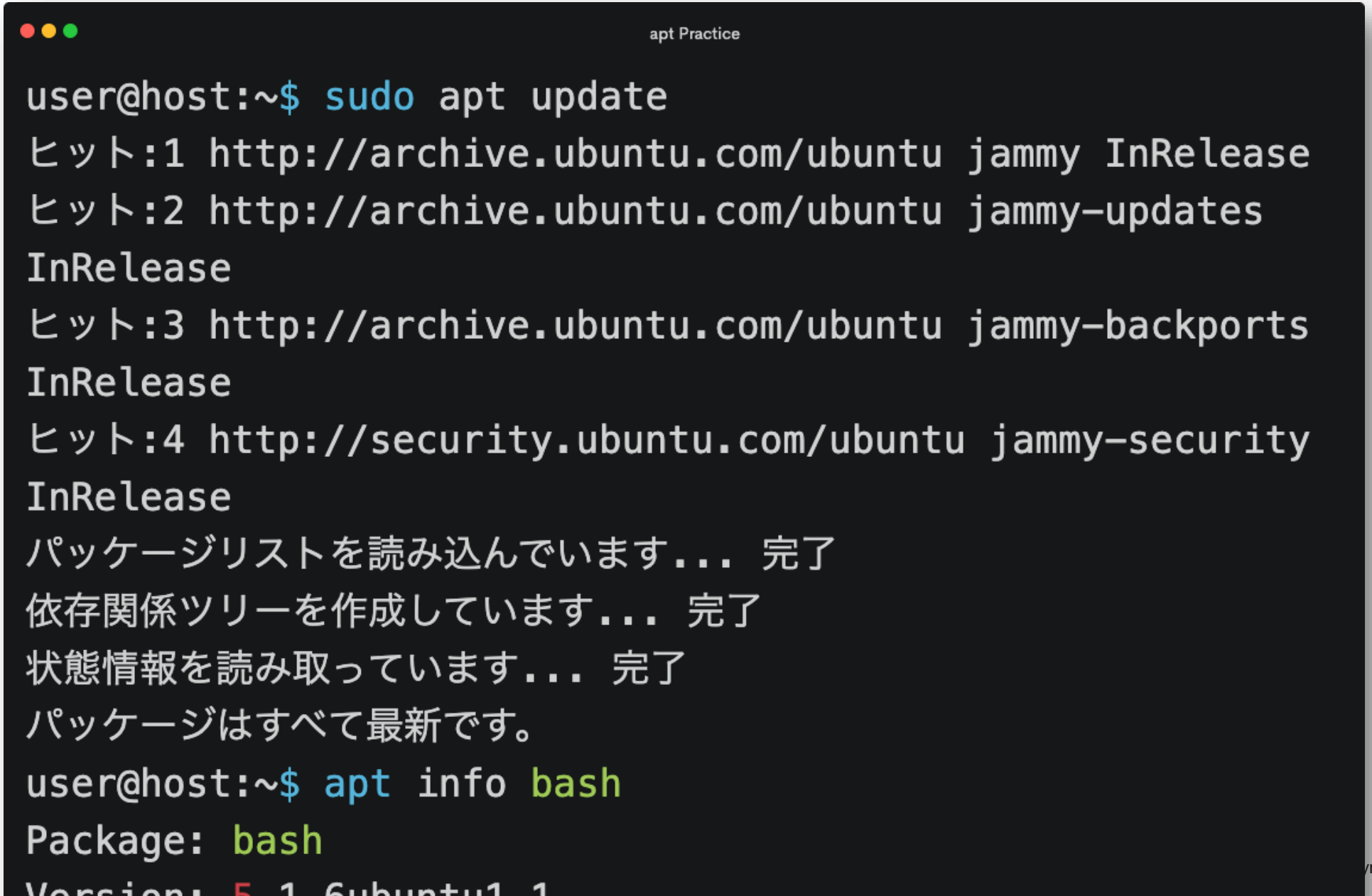
1. aptのリストを更新する
2. slパッケージの詳細を確認する
3. slパッケージをインストールする
4. slコマンドを実行する
5. slパッケージをアンインストールする

パッケージ：演習

apt Practice

```
user@host:~$ sudo apt update
ヒット:1 http://archive.ubuntu.com/ubuntu jammy InRelease
ヒット:2 http://archive.ubuntu.com/ubuntu jammy-updates
InRelease
ヒット:3 http://archive.ubuntu.com/ubuntu jammy-backports
InRelease
ヒット:4 http://security.ubuntu.com/ubuntu jammy-security
InRelease
パッケージリストを読み込んでいます... 完了
依存関係ツリーを作成しています... 完了
状態情報を読み取っています... 完了
パッケージはすべて最新です。
user@host:~$ apt info bash
Package: bash
Version: 5.1-6ubuntu1.1
Priority: required
Essential: yes
Section: shells
Origin: Ubuntu
Maintainer: Ubuntu Developers <ubuntu-devel-
discuss@lists.ubuntu.com>
Original-Maintainer: Matthias Klose <doko@debian.org>
. . .
user@host:~$ sudo apt install sl
パッケージリストを読み込んでいます... 完了
依存関係ツリーを作成しています... 完了
状態情報を読み取っています... 完了
以下のパッケージが新たにインストールされます:
  sl
アップグレード: 0 個、新規インストール: 1 個、削除: 0 個、保留: 0
個。
. . .
user@host:~$ sl
user@host:~$ sudo apt remove sl
パッケージリストを読み込んでいます... 完了
依存関係ツリーを作成しています... 完了
状態情報を読み取っています... 完了
以下のパッケージは「削除」されます:
  sl
. . .
```

パッケージ：演習



パッケージ：演習

```
user@host:~$ apt info bash
Package: bash
Version: 5.1-6ubuntu1.1
Priority: required
Essential: yes
Section: shells
Origin: Ubuntu
Maintainer: Ubuntu Developers <ubuntu-devel-
discuss@lists.ubuntu.com>
Original-Maintainer: Matthias Klose <doko@debian.org>
. . .
user@host:~$ sudo apt install sl
パッケージリストを読み込んでいます... 完了
依存関係ツリーを作成しています... 完了
状態情報を読み取っています... 完了
以下のパッケージが新たにインストールされます:
```

パッケージ：演習

```
user@host:~$ sudo apt install sl
パッケージリストを読み込んでいます... 完了
依存関係ツリーを作成しています... 完了
状態情報を読み取っています... 完了
以下のパッケージが新たにインストールされます:
  sl
アップグレード: 0 個、新規インストール: 1 個、削除: 0 個、保留: 0
個。
. . .
user@host:~$ sl
user@host:~$ sudo apt remove sl
パッケージリストを読み込んでいます... 完了
依存関係ツリーを作成しています... 完了
状態情報を読み取っています... 完了
以下のパッケージは「削除」されます:
  sl
```

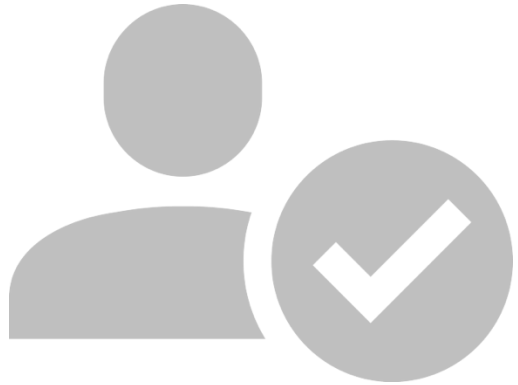
パッケージ：演習

```
. . .
user@host:~$ sl
user@host:~$ sudo apt remove sl
パッケージリストを読み込んでいます... 完了
依存関係ツリーを作成しています... 完了
状態情報を読み取っています... 完了
以下のパッケージは「削除」されます:
  sl
. . .
```

基本操作

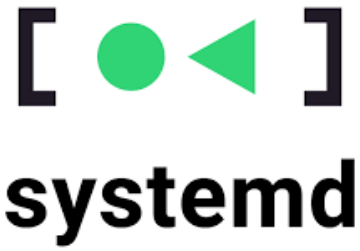
復習

ユーザ・グループ



権限

パッケージ



サービス

プロセス

- Linuxシステムの起動プロセスを制御し、
  システム全体のサービスを管理するためのソフトウェア

- 現在の主要なLinuxディストリビューション（CentOS, Ubuntu, Debianなど）
  で標準的に採用

systemd：機能と特徴

systemctlによるサービス管理
サービスの起動・停止・再起動・状態確認・自動起動設定などが可能

依存関係の自動解決
「サービスAはサービスBの後に起動する」といった依存関係を
ユニットファイルで定義し、自動で解決

ログ管理
- journaldデーモンがシステム全体のログを収集・管理
- journalctlコマンドにより、ログを時系列で簡単に確認可能

systemctlコマンド

systemdで管理されている
サービスやその他のUnit（タイマー、ソケットなど）を操作する

systemctlコマンド使用法

コマンド	内容
systemctl status <unit>	状態確認
systemctl start <unit>	起動
systemctl stop <unit>	停止
systemctl restart <unit>	再起動
systemctl enable <unit>	自動起動設定
systemctl disable <unit>	自動起動解除

Systemd:演習

1. cronサービスの状態を確認する
2. cronサービスを停止する
3. cronサービスを起動する
4. cronサービスを再起動する

サービス状態確認

sudo systemctl status cron

```
haras@hs-laptop:~$ sudo systemctl status cron
● cron.service - Regular background program processing daemon
     Loaded: loaded (/usr/lib/systemd/system/cron.service; enabled; pres>
     Active: active (running) since Fri 2025-08-08 07:38:19 JST; 45s ago
       Docs: man:cron(8)
   Main PID: 206 (cron)
      Tasks: 1 (limit: 9224)
     Memory: 436.0K (peak: 1.5M)
        CPU: 6ms
     CGroup: /system.slice/cron.service
             └─206 /usr/sbin/cron -f -P

Aug 08 07:38:19 hs-laptop systemd[1]: Started cron.service - Regular bac>
Aug 08 07:38:19 hs-laptop (cron)[206]: cron.service: Referenced but unse>
```

サービス停止

sudo systemctl stop cron

sudo systemctl status cron

```
haras@hs-laptop:~$ sudo systemctl stop cron
haras@hs-laptop:~$ sudo systemctl status cron
○ cron.service - Regular background program processing daemon
     Loaded: loaded (/usr/lib/systemd/system/cron.service; enabled; pres>
     Active: inactive (dead) since Fri 2025-08-08 07:39:48 JST; 3s ago
   Duration: 1min 29.166s
       Docs: man:cron(8)
    Process: 206 ExecStart=/usr/sbin/cron -f -P $EXTRA_OPTS (code=killed>
   Main PID: 206 (code=killed, signal=TERM)
        CPU: 7ms

Aug 08 07:38:19 hs-laptop systemd[1]: Started cron.service - Regular bac>
Aug 08 07:38:19 hs-laptop (cron)[206]: cron.service: Referenced but unse>
```

サービス起動

sudo systemctl start cron

sudo systemctl status cron

```
haras@hs-laptop:~$ sudo systemctl start cron
haras@hs-laptop:~$ sudo systemctl status cron
● cron.service - Regular background program processing daemon
     Loaded: loaded (/usr/lib/systemd/system/cron.service; enabled; pres>
     Active: active (running) since Fri 2025-08-08 07:42:39 JST; 4s ago
       Docs: man:cron(8)
   Main PID: 819 (cron)
      Tasks: 1 (limit: 9224)
     Memory: 368.0K (peak: 552.0K)
        CPU: 6ms
     CGroup: /system.slice/cron.service
             └─819 /usr/sbin/cron -f -P

Aug 08 07:42:39 hs-laptop systemd[1]: Started cron.service - Regular bac>
Aug 08 07:42:39 hs-laptop cron[819]: (CRON) INFO (pidfile fd = 3)
```

サービス再起動

sudo systemctl restart cron

sudo systemctl status cron

```
haras@hs-laptop:~$ sudo systemctl restart cron
haras@hs-laptop:~$ sudo systemctl status cron
● cron.service - Regular background program processing daemon
     Loaded: loaded (/usr/lib/systemd/system/cron.service; enabled; pres>
     Active: active (running) since Fri 2025-08-08 07:44:14 JST; 5s ago
       Docs: man:cron(8)
   Main PID: 833 (cron)
      Tasks: 1 (limit: 9224)
     Memory: 344.0K (peak: 552.0K)
        CPU: 6ms
     CGroup: /system.slice/cron.service
             └─833 /usr/sbin/cron -f -P

Aug 08 07:44:14 hs-laptop systemd[1]: Started cron.service - Regular bac>
Aug 08 07:44:14 hs-laptop (cron)[833]: cron.service: Referenced but unse>
```

Unitファイル

- systemdがサービスやタスクを
  どのように管理するかを定義するための設定ファイル

- テキスト形式で記述され、
  「何を」「いつ」「どのように」実行するかを
  systemdに伝える

Unitファイルの例

ファイル場所：/etc/systemd/system/ など

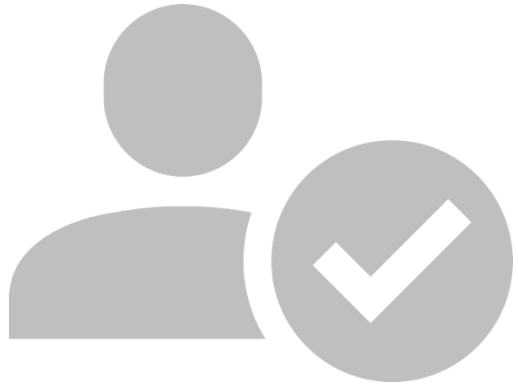
```
haru@haru-desktop:/mnt/c/Users/haras$ cat /etc/systemd/system/mult
i-user.target.wants/cron.service
[Unit]
Description=Regular background program processing daemon
Documentation=man:cron(8)
After=remote-fs.target nss-user-lookup.target

[Service]
EnvironmentFile=-/etc/default/cron
ExecStart=/usr/sbin/cron -f -P $EXTRA_OPTS
IgnoreSIGPIPE=false
KillMode=process
Restart=on-failure
SyslogFacility=cron

[Install]
WantedBy=multi-user.target
```

セクション	内容
[Unit]	説明・依存設定
[Service]	実行コマンド・ユーザ指定
[Install]	有効化設定

基本操作

復習

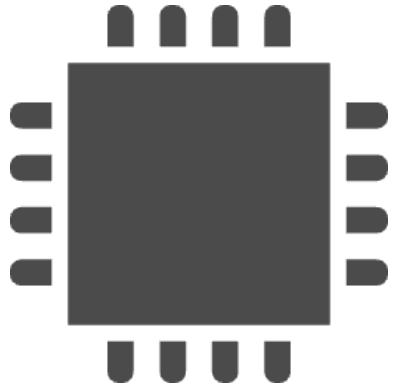
ユーザ・グループ



権限

パッケージ

サービス



プロセス

プロセス

- コンピュータ上で実行されているプログラム
- Linuxで動くプログラムの一つ一つがプロセス
- OSが管理する単位

ジョブ

- プロセスの集まりがジョブ
- シェルが管理する単位

PIDとPPID

- コンピュータ上で動くすべてのプロセスには一意の番号が振られている
- **PID**と呼ぶ

- プロセスはプロセスを呼び出せる（親子関係）
- あるプロセスを起動した元のプロセス（親）のPIDを**PPID**と呼ぶ

プロセスツリー

```
haru@haru-desktop:~$ pstree
systemd─┬─2*[agetty]
        ├─cron
        ├─dbus-daemon
        ├─init-systemd(Ub─┬─SessionLeader───Relay(686)───bash───pstree
        │                 ├─init───{init}
        │                 ├─login───bash
        │                 └─{init-systemd(Ub}
        ├─polkitd───3*[{polkitd}]
        ├─rsyslogd───3*[{rsyslogd}]
        ├─systemd───(sd-pam)
        ├─systemd-journal
        ├─systemd-logind
        ├─systemd-resolve
        ├─systemd-timesyn───{systemd-timesyn}
        ├─systemd-udevd
        ├─unattended-upgr───{unattended-upgr}
        └─wsl-pro-service───7*[{wsl-pro-service}]
```

コンピュータで動くプロセスは全て親子関係を持つ
ツリー構造を形成している

ps (process status) コマンド

```
haru@haru-desktop:~$ ps aux
USER         PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root           1  0.0  0.0  21780 12524 ?        Ss   10:38   0:00 /sbin/init
root           2  0.0  0.0   3060  1664 ?        Sl   10:38   0:00 /init
root           7  0.0  0.0   3076  1792 ?        Sl   10:38   0:00 plan9 --control-socket 7 --log-level 4 --server
root          59  0.0  0.1  66820 18756 ?        S<s  10:38   0:00 /usr/lib/systemd/systemd-journald
root         109  0.0  0.0  25400  6272 ?        Ss   10:38   0:00 /usr/lib/systemd/systemd-udevd
systemd+     120  0.0  0.0  21452 12544 ?        Ss   10:38   0:00 /usr/lib/systemd/systemd-resolved
systemd+     121  0.0  0.0  91020  7680 ?        Ssl  10:38   0:00 /usr/lib/systemd/systemd-timesyncd
message+     185  0.0  0.0   9588  4992 ?        Ss   10:38   0:00 @dbus-daemon --system --address=systemd: --nofo
root         195  0.0  0.0  17956  8448 ?        Ss   10:38   0:00 /usr/lib/systemd/systemd-logind
root         197  0.0  0.0 1756096 12416 ?       Ssl  10:38   0:00 /usr/libexec/wsl-pro-service -vv
root         200  0.0  0.0   3160  1920 hvc0     Ss+  10:38   0:00 /sbin/agetty -o -p -- \u --noclear --keep-baud
root         216  0.0  0.0   3116  1792 tty1     Ss+  10:38   0:00 /sbin/agetty -o -p -- \u --noclear - linux
syslog       217  0.0  0.0 222508  5504 ?        Ssl  10:38   0:00 /usr/sbin/rsyslogd -n -iNONE
```

- 実行されているプロセスの一覧（スナップショット）を表示
- サーバの状況確認や特定のプロセスを探す時、
  親プロセスを確認する時に用いる

killコマンドとpkillコマンド

- プロセスを終了させるコマンド
- OSからプロセスに、SIGTERMシグナル（終了を要望）や、SIGKILLシグナル（強制終了）を送ることにより、プロセスを終わらせることが出来る。
- killはPIDでプロセスを指定して終了
- pkillは指定したプロセス名に当てはまるプロセスをすべて終了する

```
haru@haru-desktop:~$ sleep 600 &
[1] 6254
haru@haru-desktop:~$ kill 6254
```

```
haru@haru-desktop:~$ sleep 600 &
[3] 6259
haru@haru-desktop:~$ pkill sleep
[2]-  Terminated              sleep 600
[3]+  Terminated              sleep 600
```

プロセス:演習

1. sleepプロセスを作成する
2. 作成したプロセスの情報を確認する
3. pstreeでプロセスの親子関係を確認する
4. プロセスをkillで終了する
5. プロセスが削除されたことを確認する

sleepプロセスの作成

Sleep 600 &

```
haras@hs-laptop:~$ sleep 600 &
[1] 1070
```

sleepプロセスの情報確認

ps aux | grep sleep

```
haras@hs-laptop:~$ ps aux | grep sleep
haras        1070  0.0  0.0   3124  1584 pts/0    S    08:19   0:00 sleep 600
haras        1074  0.0  0.0   4088  1760 pts/0    S+   08:19   0:00 grep --colo
r=auto sleep
```

sleepプロセスの親子関係を確認する

$ sudo apt install psmisc

pstree

```
haras@hs-laptop:~$ pstree
systemd─┬─NetworkManager───3*[{NetworkManager}]
        ├─accounts-daemon───3*[{accounts-daemon}]
        ├─agetty
        ├─avahi-daemon───avahi-daemon
        ├─cron
        ├─dbus-daemon
        ├─gdm3───3*[{gdm3}]
        ├─gnome-remote-de───3*[{gnome-remote-de}]
        ├─init-systemd(Ub─┬─SessionLeader───Relay(1050)───bash─┬─pstree
        │                 │                                  └─sleep
        │                 ├─init───{init}
        │                 └─login───bash
```

sleepプロセスを終了させる

kill "確認したPID"

```
haras@hs-laptop:~$ kill 1070
```

sleepプロセスを終了したことを確認

ps aux | grep sleep

```
haras@hs-laptop:~$ ps aux | grep sleep
haras        1078  0.0  0.0   4088  1760 pts/0    S+   08:21   0:00 grep --colo
r=auto sleep
[1]+  Terminated              sleep 600
```

topコマンドとhtopコマンド

- システムのCPU使用率やメモリ使用率をプロセスごとに表示したりソートしてリアルタイムで表示したりできるソフトウェア
- htopはtopをよりグラフィカルにしたアプリケーション

top

```
top - 11:52:05 up  1:13,  1 user,  load average: 0.15, 0.03, 0.01
Tasks:  25 total,   1 running,  24 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.0 us,  0.0 sy,  0.0 ni, 99.9 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
MiB Mem :  13908.2 total,  13223.7 free,    582.9 used,    286.6 buff/cache
MiB Swap:   4096.0 total,   4096.0 free,      0.0 used.  13325.4 avail Mem

    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
      1 root      20   0   21780  12524   9324 S   0.0   0.1   0:00.99 systemd
      2 root      20   0    3060   1664   1664 S   0.0   0.0   0:00.06 init-systemd(Ub
      7 root      20   0    3076   1792   1792 S   0.0   0.0   0:00.00 init
     59 root      19  -1   66820  18756  17860 S   0.0   0.1   0:00.45 systemd-journal
    109 root      20   0   25400   6272   4864 S   0.0   0.0   0:00.22 systemd-udevd
    120 systemd+  20   0   21452  12544  10368 S   0.0   0.1   0:00.20 systemd-resolve
    121 systemd+  20   0   91020   7680   6912 S   0.0   0.1   0:00.17 systemd-timesyn
    185 message+  20   0    9588   4992   4480 S   0.0   0.0   0:00.16 dbus-daemon
    195 root      20   0   17956   8448   7552 S   0.0   0.1   0:00.17 systemd-logind
    197 root      20   0 1756096  12416  10368 S   0.0   0.1   0:00.23 wsl-pro-service
    200 root      20   0    3160   1920   1792 S   0.0   0.0   0:00.00 agetty
    216 root      20   0    3116   1792   1664 S   0.0   0.0   0:00.01 agetty
    217 syslog    20   0  222508   5504   4480 S   0.0   0.0   0:00.14 rsyslogd
    225 root      20   0  107016  22272  13184 S   0.0   0.2   0:00.18 unattended-upgr
```

htop

```
  0[                                        0.0%]   4[                                        0.0%]
  1[                                        0.0%]   5[                                        0.0%]
  2[                                        0.0%]   6[                                        0.0%]
  3[                                        0.0%]   7[                                        0.0%]
Mem[|||||                           402M/13.6G] Tasks: 25, 20 thr, 0 kthr; 1 running
Swp[                                  0K/4.00G] Load average: 0.12 0.03 0.01
                                                Uptime: 01:13:57

  Main   I/O
    PID USER       PRI  NI  VIRT   RES   SHR S  CPU%▽MEM%   TIME+  Command
      1 root        20   0 21780 12524  9324 S   0.0  0.1  0:01.00 /sbin/init
      2 root        20   0  3060  1664  1664 S   0.0  0.0  0:00.05 /init
      7 root        20   0  3076  1792  1792 S   0.0  0.0  0:00.00 plan9 --control-socket 7 --log-level 4 --server
      8 root        20   0  3076  1792  1792 S   0.0  0.0  0:00.00 plan9 --control-socket 7 --log-level 4 --server
      9 root        20   0  3060  1664  1664 S   0.0  0.0  0:00.00 /init
     59 root        19  -1 66820 18756 17860 S   0.0  0.1  0:00.38 /usr/lib/systemd/systemd-journald
    109 root        20   0 25400  6272  4864 S   0.0  0.0  0:00.22 /usr/lib/systemd/systemd-udevd
    120 systemd-re  20   0 21452 12544 10368 S   0.0  0.1  0:00.20 /usr/lib/systemd/systemd-resolved
    121 systemd-ti  20   0 91020  7680  6912 S   0.0  0.1  0:00.17 /usr/lib/systemd/systemd-timesyncd
F1Help  F2Setup F3SearchF4FilterF5Tree  F6SortByF7Nice -F8Nice +F9Kill  F10Quit
```

サーバモニタリング:演習

top

```
top - 11:52:05 up  1:13,  1 user,  load average: 0.15, 0.03, 0.01
Tasks:  25 total,   1 running,  24 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.0 us,  0.0 sy,  0.0 ni, 99.9 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
MiB Mem :  13908.2 total,  13223.7 free,    582.9 used,    286.6 buff/cache
MiB Swap:   4096.0 total,   4096.0 free,      0.0 used.  13325.4 avail Mem

    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
      1 root      20   0   21780  12524   9324 S   0.0   0.1   0:00.99 systemd
      2 root      20   0    3060   1664   1664 S   0.0   0.0   0:00.06 init-systemd(Ub
      7 root      20   0    3076   1792   1792 S   0.0   0.0   0:00.00 init
     59 root      19  -1   66820  18756  17860 S   0.0   0.1   0:00.45 systemd-journal
    109 root      20   0   25400   6272   4864 S   0.0   0.0   0:00.22 systemd-udevd
    120 systemd+  20   0   21452  12544  10368 S   0.0   0.1   0:00.20 systemd-resolve
    121 systemd+  20   0   91020   7680   6912 S   0.0   0.1   0:00.17 systemd-timesyn
    185 message+  20   0    9588   4992   4480 S   0.0   0.0   0:00.16 dbus-daemon
    195 root      20   0   17956   8448   7552 S   0.0   0.1   0:00.17 systemd-logind
    197 root      20   0 1756096  12416  10368 S   0.0   0.1   0:00.23 wsl-pro-service
    200 root      20   0    3160   1920   1792 S   0.0   0.0   0:00.00 agetty
    216 root      20   0    3116   1792   1664 S   0.0   0.0   0:00.01 agetty
    217 syslog    20   0  222508   5504   4480 S   0.0   0.0   0:00.14 rsyslogd
    225 root      20   0  107016  22272  13184 S   0.0   0.2   0:00.18 unattended-upgr
```

sudo apt install htop

htop

```
  0[                                                    0.0%]   4[                                                    0.0%]
  1[                                                    0.0%]   5[                                                    0.0%]
  2[                                                    0.0%]   6[                                                    0.0%]
  3[                                                    0.0%]   7[                                                    0.0%]
Mem[|||||                                         402M/13.6G] Tasks: 25, 20 thr, 0 kthr; 1 running
Swp[                                                0K/4.00G] Load average: 0.12 0.03 0.01
                                                              Uptime: 01:13:57

  Main   I/O
    PID USER       PRI  NI  VIRT   RES   SHR S  CPU%▽MEM%   TIME+  Command
      1 root        20   0 21780 12524  9324 S   0.0  0.1  0:01.00 /sbin/init
      2 root        20   0  3060  1664  1664 S   0.0  0.0  0:00.05 /init
      7 root        20   0  3076  1792  1792 S   0.0  0.0  0:00.00 plan9 --control-socket 7 --log-level 4 --server
      8 root        20   0  3076  1792  1792 S   0.0  0.0  0:00.00 plan9 --control-socket 7 --log-level 4 --server
      9 root        20   0  3060  1664  1664 S   0.0  0.0  0:00.00 /init
     59 root        19  -1 66820 18756 17860 S   0.0  0.1  0:00.38 /usr/lib/systemd/systemd-journald
    109 root        20   0 25400  6272  4864 S   0.0  0.0  0:00.22 /usr/lib/systemd/systemd-udevd
    120 systemd-re  20   0 21452 12544 10368 S   0.0  0.1  0:00.20 /usr/lib/systemd/systemd-resolved
    121 systemd-ti  20   0 91020  7680  6912 S   0.0  0.1  0:00.17 /usr/lib/systemd/systemd-timesyncd
F1Help  F2Setup F3SearchF4FilterF5Tree  F6SortByF7Nice -F8Nice +F9Kill  F10Quit
```

journalctl

- systemdが収集・管理するログ（ジャーナル）を閲覧・検索するためのコマンド
- ログを一元的に管理し、時間、サービス、優先度などで検索出来る

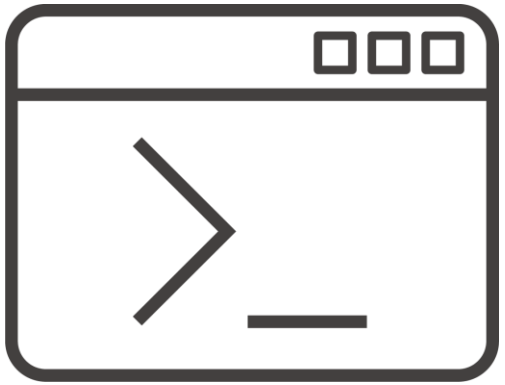
```
haru@haru-desktop:~$ journalctl
Aug 02 10:40:08 haru-desktop kernel: Linux version 6.6.87.2-microsoft-standard-WSL2 (root@439a258ad544) (gcc (GCC>
Aug 02 10:40:08 haru-desktop kernel: Command line: initrd=\initrd.img WSL_ROOT_INIT=1 panic=-1 nr_cpus=8 hv_utils>
Aug 02 10:40:08 haru-desktop kernel: KERNEL supported cpus:
Aug 02 10:40:08 haru-desktop kernel:   Intel GenuineIntel
Aug 02 10:40:08 haru-desktop kernel:   AMD AuthenticAMD
Aug 02 10:40:08 haru-desktop kernel: BIOS-provided physical RAM map:
Aug 02 10:40:08 haru-desktop kernel: BIOS-e820: [mem 0x0000000000000000-0x000000000009ffff] usable
Aug 02 10:40:08 haru-desktop kernel: BIOS-e820: [mem 0x00000000000e0000-0x00000000000e0fff] reserved
Aug 02 10:40:08 haru-desktop kernel: BIOS-e820: [mem 0x0000000000100000-0x00000000001fffff] ACPI data
Aug 02 10:40:08 haru-desktop kernel: BIOS-e820: [mem 0x0000000000200000-0x00000000f7ffffff] usable
Aug 02 10:40:08 haru-desktop kernel: BIOS-e820: [mem 0x0000000100000000-0x00000003839fffff] usable
Aug 02 10:40:08 haru-desktop kernel: NX (Execute Disable) protection: active
Aug 02 10:40:08 haru-desktop kernel: APIC: Static calls initialized
Aug 02 10:40:08 haru-desktop kernel: DMI not present or invalid.
Aug 02 10:40:08 haru-desktop kernel: Hypervisor detected: Microsoft Hyper-V
Aug 02 10:40:08 haru-desktop kernel: Hyper-V: privilege flags low 0x2e7f, high 0x3b8030, hints 0x924c2c, misc 0xe>
Aug 02 10:40:08 haru-desktop kernel: Hyper-V: Nested features: 0x3e0101
```

journalctlでcron関連のログを調べる

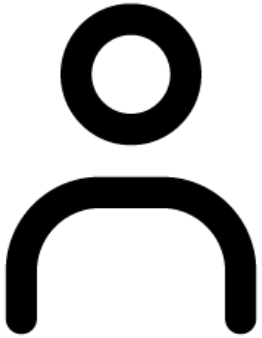
journalctl -u cron

```
haras@hs-laptop:~$ journalctl -u cron
Jul 30 08:24:45 hs-laptop systemd[1]: Started cron.service - Regular bac>
Jul 30 08:24:45 hs-laptop (cron)[205]: cron.service: Referenced but unse>
Jul 30 08:24:45 hs-laptop cron[205]: (CRON) INFO (pidfile fd = 3)
Jul 30 08:24:45 hs-laptop cron[205]: (CRON) INFO (Running @reboot jobs)
-- Boot de962baebffc423290ab6ad10ce442ea --
Jul 31 15:09:17 hs-laptop systemd[1]: Started cron.service - Regular bac>
Jul 31 15:09:17 hs-laptop (cron)[197]: cron.service: Referenced but unse>
Jul 31 15:09:17 hs-laptop cron[197]: (CRON) INFO (pidfile fd = 3)
Jul 31 15:09:17 hs-laptop cron[197]: (CRON) INFO (Running @reboot jobs)
Jul 31 15:17:01 hs-laptop CRON[15754]: pam_unix(cron:session): session o>
Jul 31 15:17:01 hs-laptop CRON[15755]: (root) CMD (cd / && run-parts --r>
Jul 31 15:17:01 hs-laptop CRON[15754]: pam_unix(cron:session): session c>
-- Boot 92412acde09f4b42bbb028bb282ce994 --
```

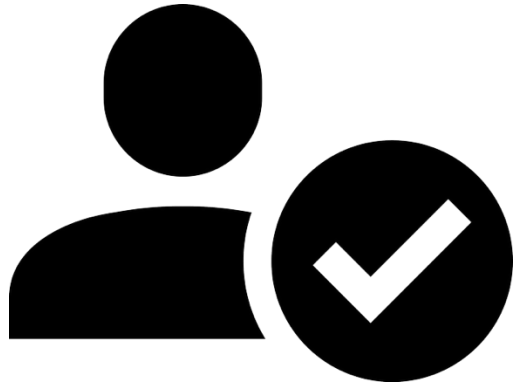
基本操作



復習



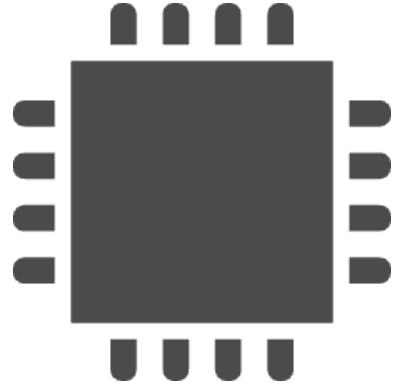
ユーザ・グループ



権限

パッケージ

サービス



プロセス

ネットワークについて

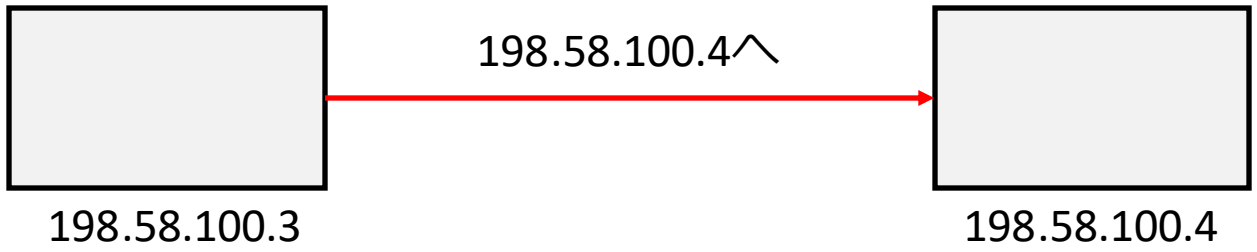
1. IPアドレス
2. 名前解決
3. ルータ
4. ポート

ネットワーク

コンピュータ同士が情報をやり取りする仕組みや構造

IPアドレス

ネットワーク上で機器を識別するための番号



ポイント

- 機器同士が通信するにはお互いのIPアドレスが必要
- グローバルIPアドレス: 世界で一意
- ローカルIPアドレス: ネットワーク内で一意
- ipコマンドで確認できる

自分のIPアドレスを確認してみよう

ip addr

```
haras@hs-laptop:~$ ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group defa
ult qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet 10.255.255.254/32 brd 10.255.255.254 scope global lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group de
fault qlen 1000
    link/ether 00:15:5d:b3:89:4e brd ff:ff:ff:ff:ff:ff
    inet 172.18.212.52/20 brd 172.18.223.255 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::215:5dff:feb3:894e/64 scope link
       valid_lft forever preferred_lft forever
```

名前解決

ホスト名やドメイン名とIPアドレス間を変換する処理

例

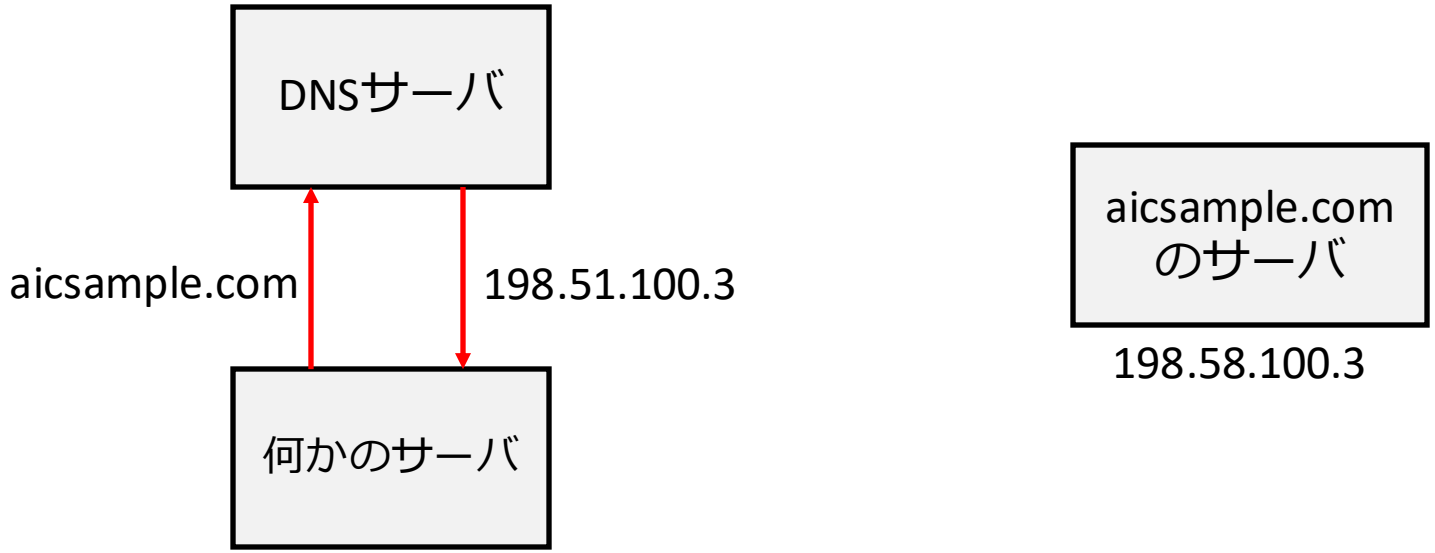
- aicsample.com -> 198.58.100.3
- 198.58.100.3 -> aicsample.com
- https://google.com -> https://142.250.190.78
- ホスト名: pc1, server1など
- ドメイン名: www.google.com, example.com など

DNS (Domain Name System)

名前解決を実現するための分散型データベースシステム

ポイント

- 名前解決をする際はDNSサーバに聞いてIPアドレスを教えてもらう
- aicsample.com のIPアドレスを教えて -> 198.58.100.3 です



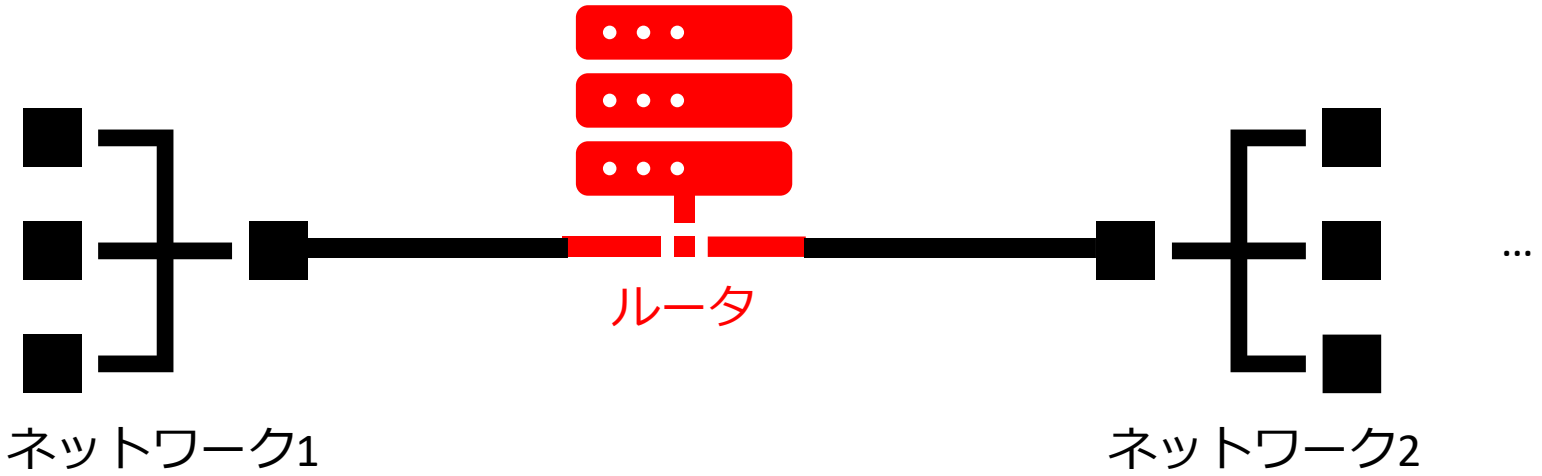
ドメイン名からIPアドレスを求める

sudo apt install bind9-dnsutils
nslookup aic.keio.ac.jp

```
haras@hs-laptop:~$ nslookup aic.keio.ac.jp
Server:         10.255.255.254
Address:        10.255.255.254#53

Non-authoritative answer:
Name:   aic.keio.ac.jp
Address: 133.242.249.28
```

ルータ

異なるネットワーク同士を接続して
データを正しい宛先に転送する機器



ポイント
- 通信先のIPアドレスを見て、次に送る先を判断
- ネットワークとネットワークの間にある

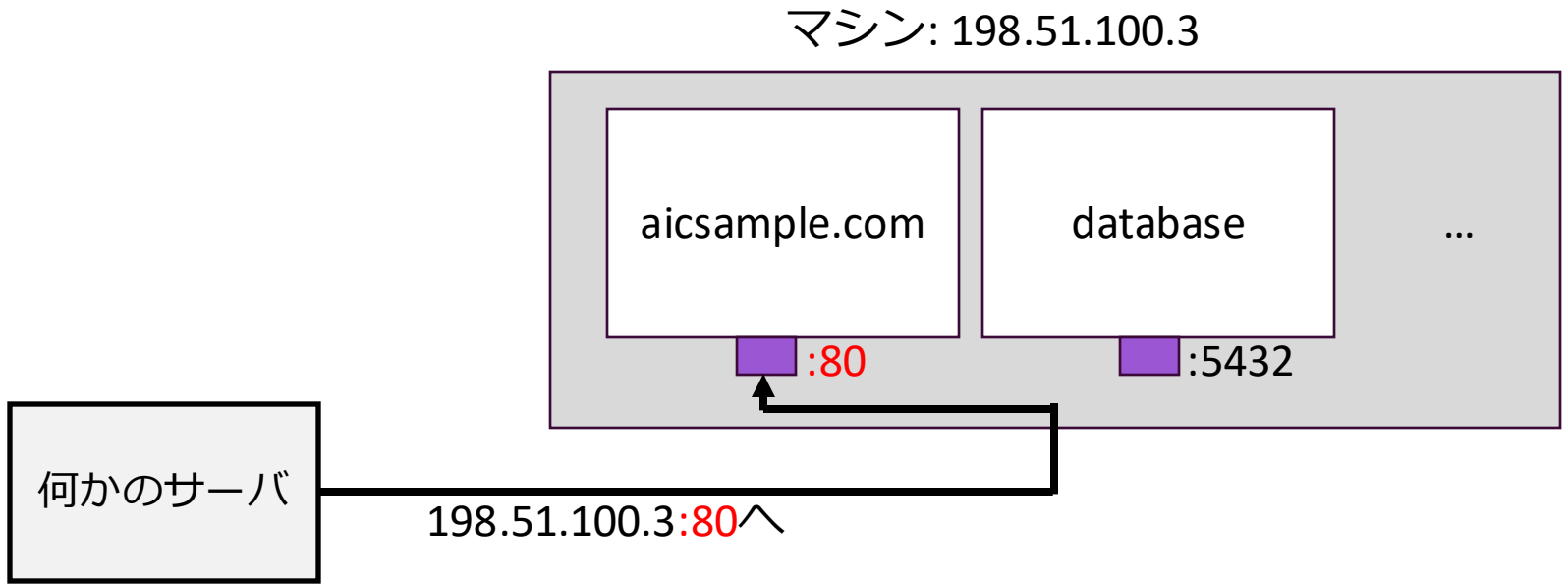
ゲートウェイ

あるネットワークから別のネットワークに出るための出入口

ポート

1つのIPアドレス上で複数の通信を区別するための番号

ポイント
- 同じIPアドレス内で複数のサービスを動かすために使う
- 通信は「IPアドレス＋ポート番号」のセットで成り立つ



telnetコマンドでサーバに接続できるか確認する

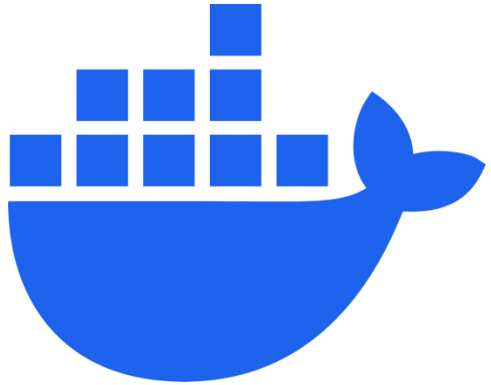
sudo apt install inetutils-telnet
telnet aic.keio.ac.jp 1234
telnet aic.keio.ac.jp 80
GET /

```
haras@hs-laptop:~$ telnet aic.keio.ac.jp 1234
Trying 133.242.249.28...
telnet: Unable to connect to remote host: Connection refused
haras@hs-laptop:~$ telnet aic.keio.ac.jp 80
Trying 133.242.249.28...
Connected to aic.keio.ac.jp.
Escape character is '^]'.
GET /
```

本日の流れ

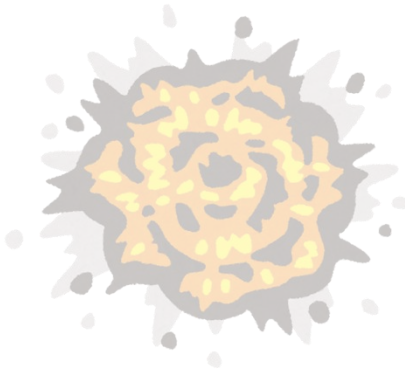
概論
「大学のサバ管」

Linux管理者のための
基本操作



Webサーバ
on Docker

トラブルシューティング



破壊

Dockerでサーバを立ててみる

沼澤貴大

サーバを立てる

**サービスを提供するためのプログラムや環境を準備して、
ネットワークからアクセスできるようにすること**

Webサーバを立てる時の例

1. Webページの中身 (HTMLなど) を準備する
2. リクエストが来た際に中身を送信するプログラムを準備する
3. ネットワーク越しにアクセスできるように設定する

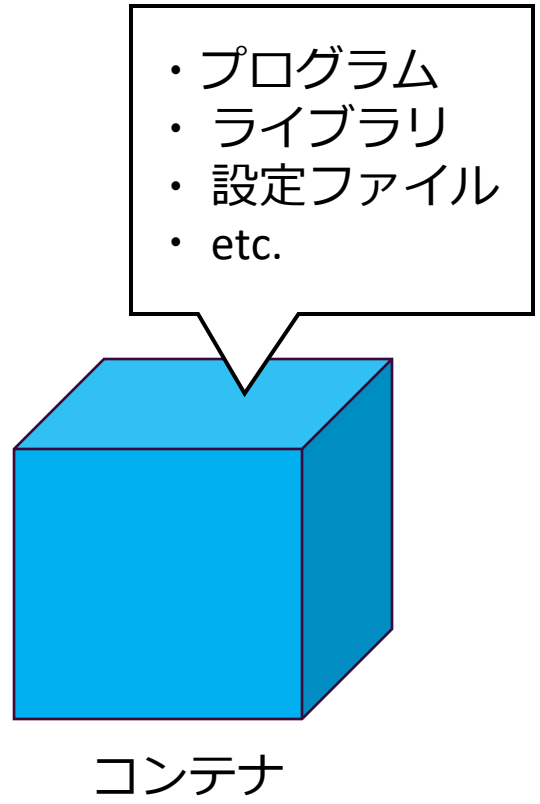
Docker
データやプログラムを隔離しながら実行できる仕組み

ポイント

- プログラムを「**コンテナ**」という隔離された箱の中で動かす
- コンテナは隔離されているため、他のシステムに影響を与えない
- どこでも同じように動く

コンテナ

プログラムとその実行環境をまとめて動かせる隔離された実行単位 (プロセス)



ポイント

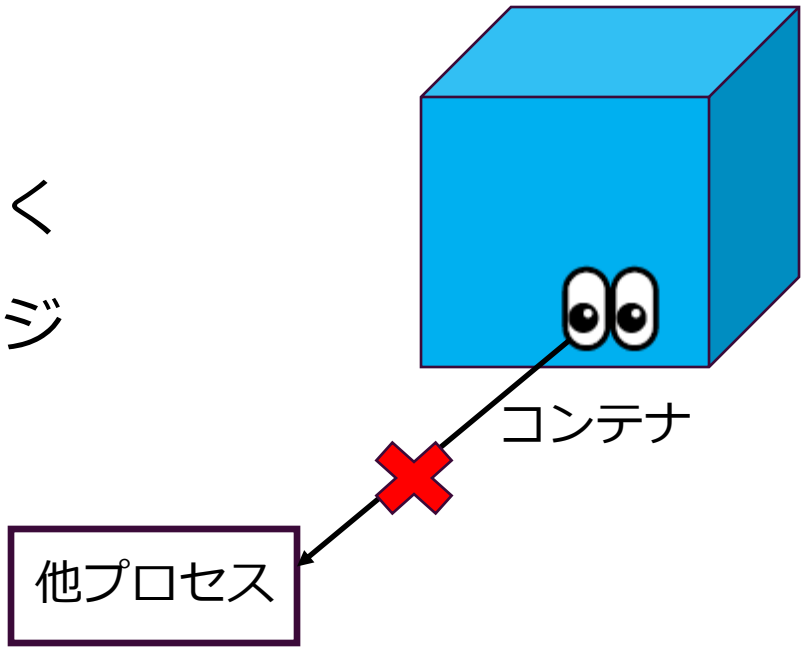
- 他のコンテナやホスト環境と分離されて動く
- 必要なライブラリや設定も一緒にパッケージ
- 仮想マシンより高速・省リソース

コンテナ

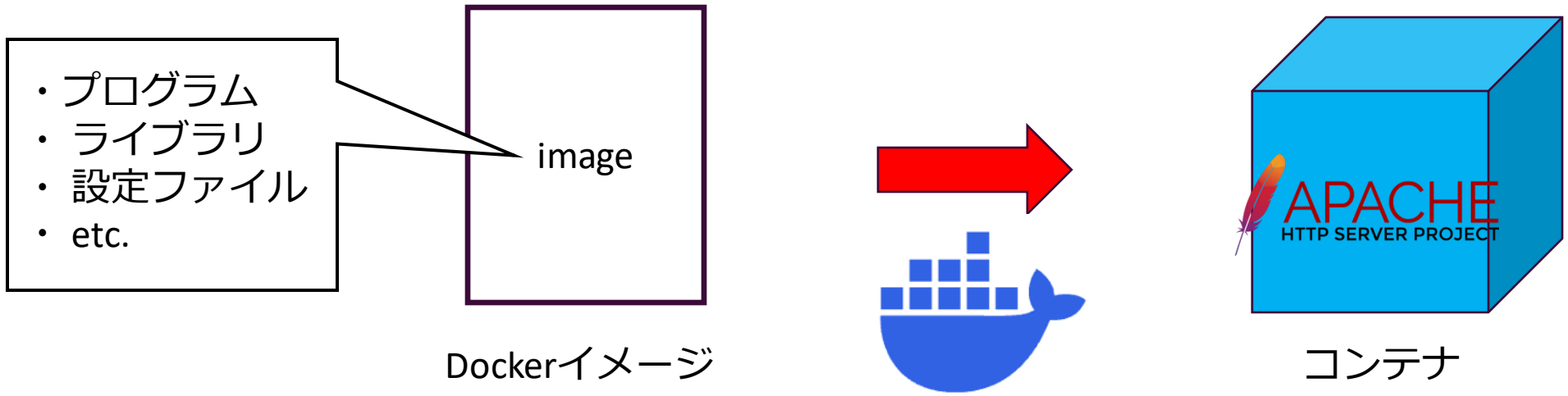
プログラムとその実行環境をまとめて動かせる
隔離された実行単位 (プロセス)

ポイント

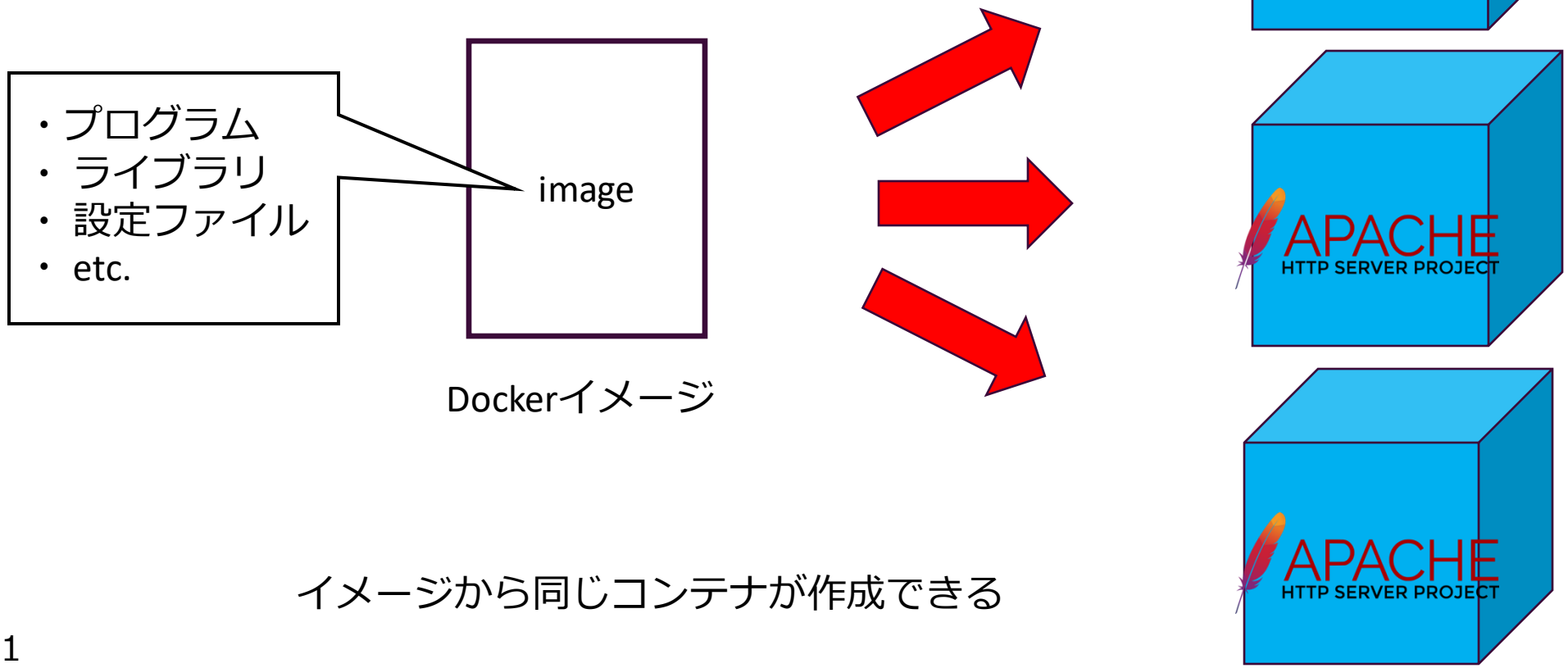
- 他のコンテナやホスト環境と分離されて動く
- 必要なライブラリや設定も一緒にパッケージ
- 仮想マシンより高速・省リソース



イメージ
コンテナの元となる設計図



イメージ
コンテナの元となる設計図

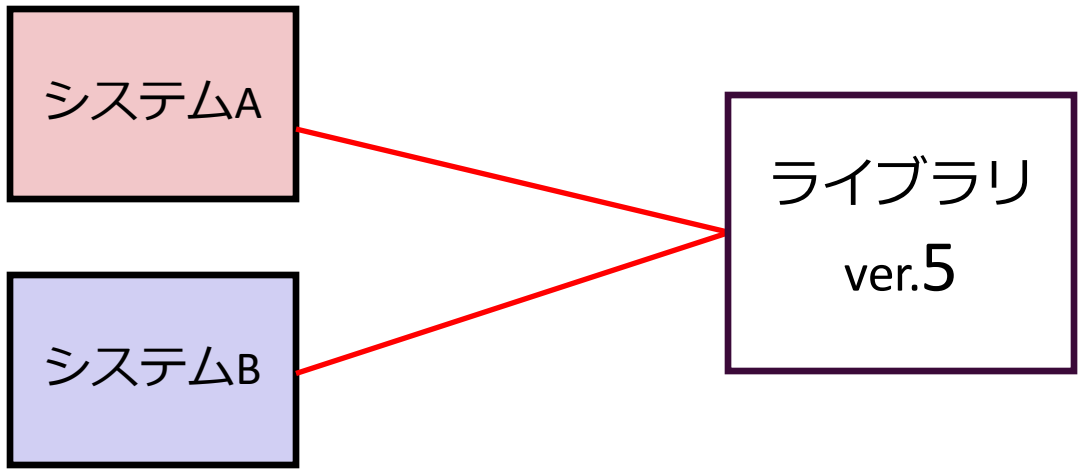


イメージから同じコンテナが作成できる

メリット1: 他のシステムに影響を与えない

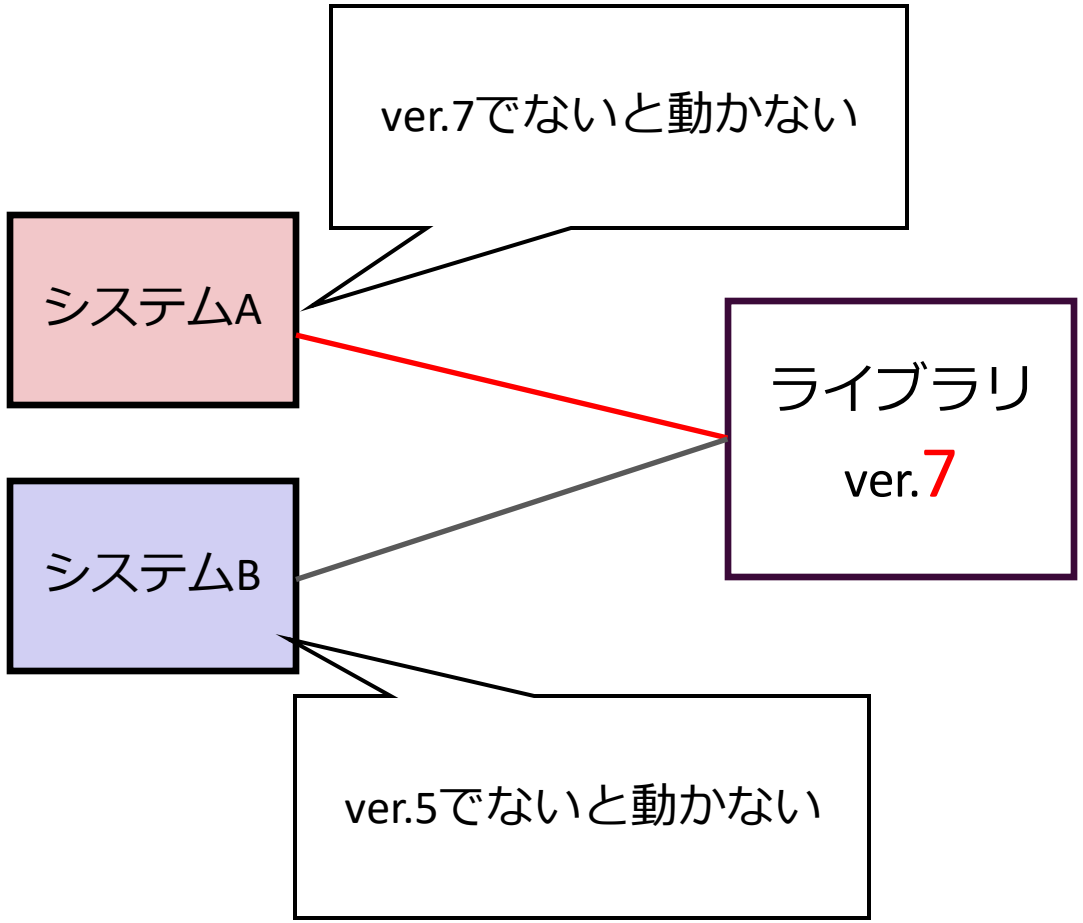
メリット1: 他のシステムに影響を与えない

コンテナを使わずに問題となるケース



メリット1: 他のシステムに影響を与えない

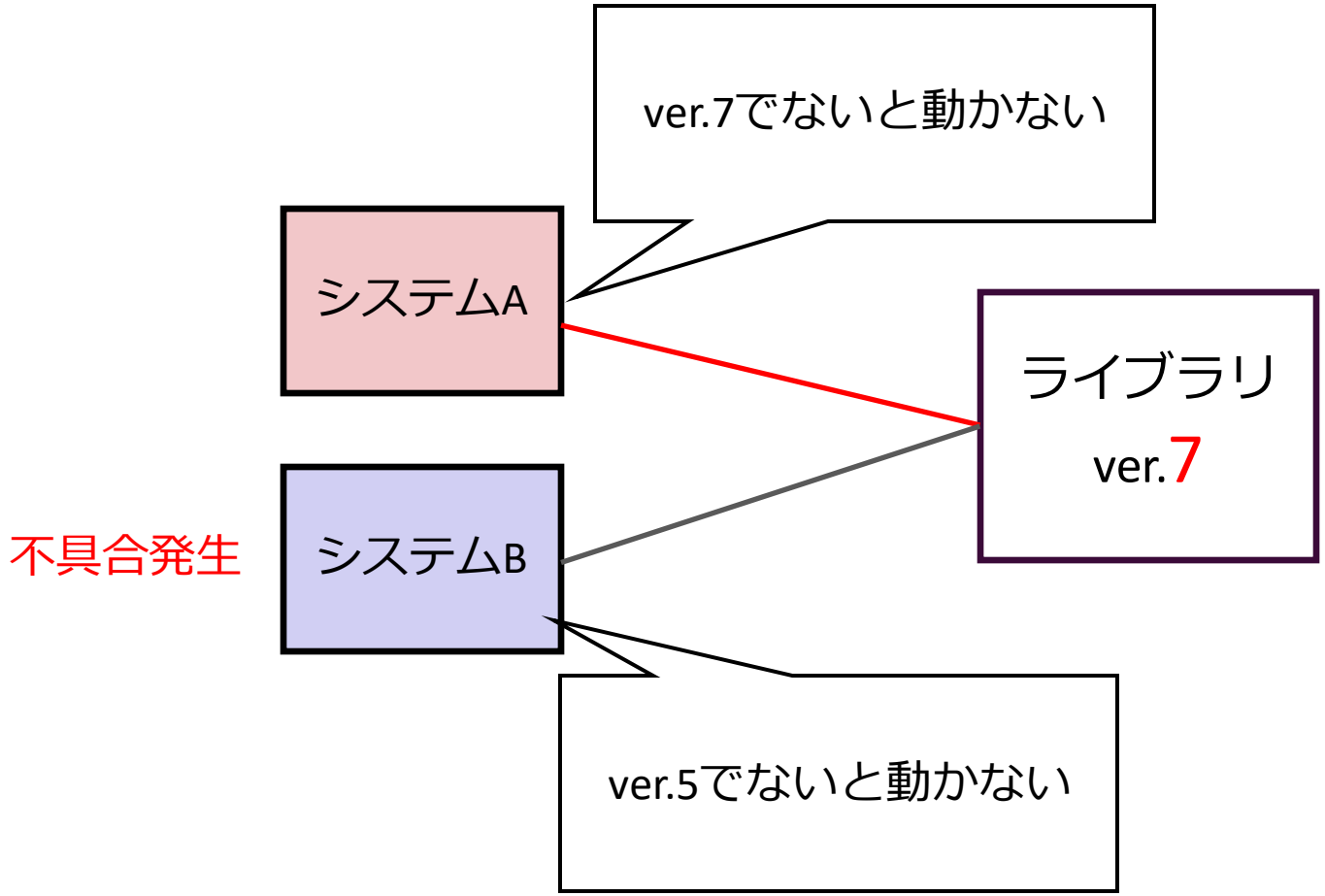
コンテナを使わずに問題となるケース



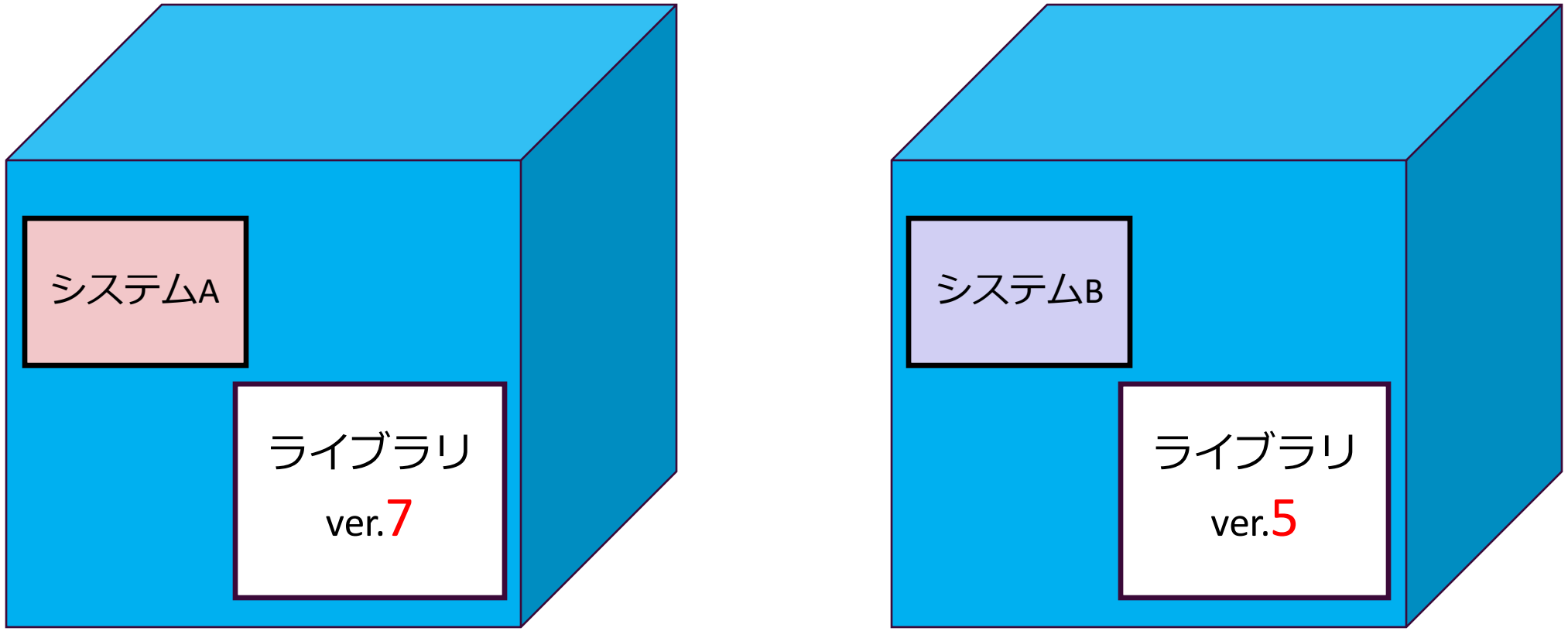
システムA用に
ver.5 -> ver.7

メリット1: 他のシステムに影響を与えない

コンテナを使わずに問題となるケース

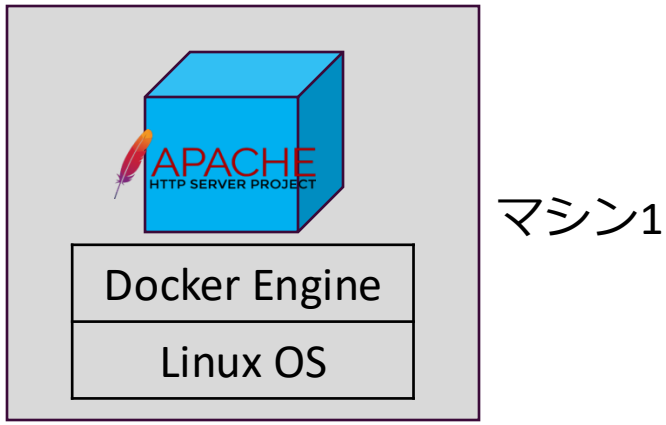


メリット1: 他のシステムに影響を与えない



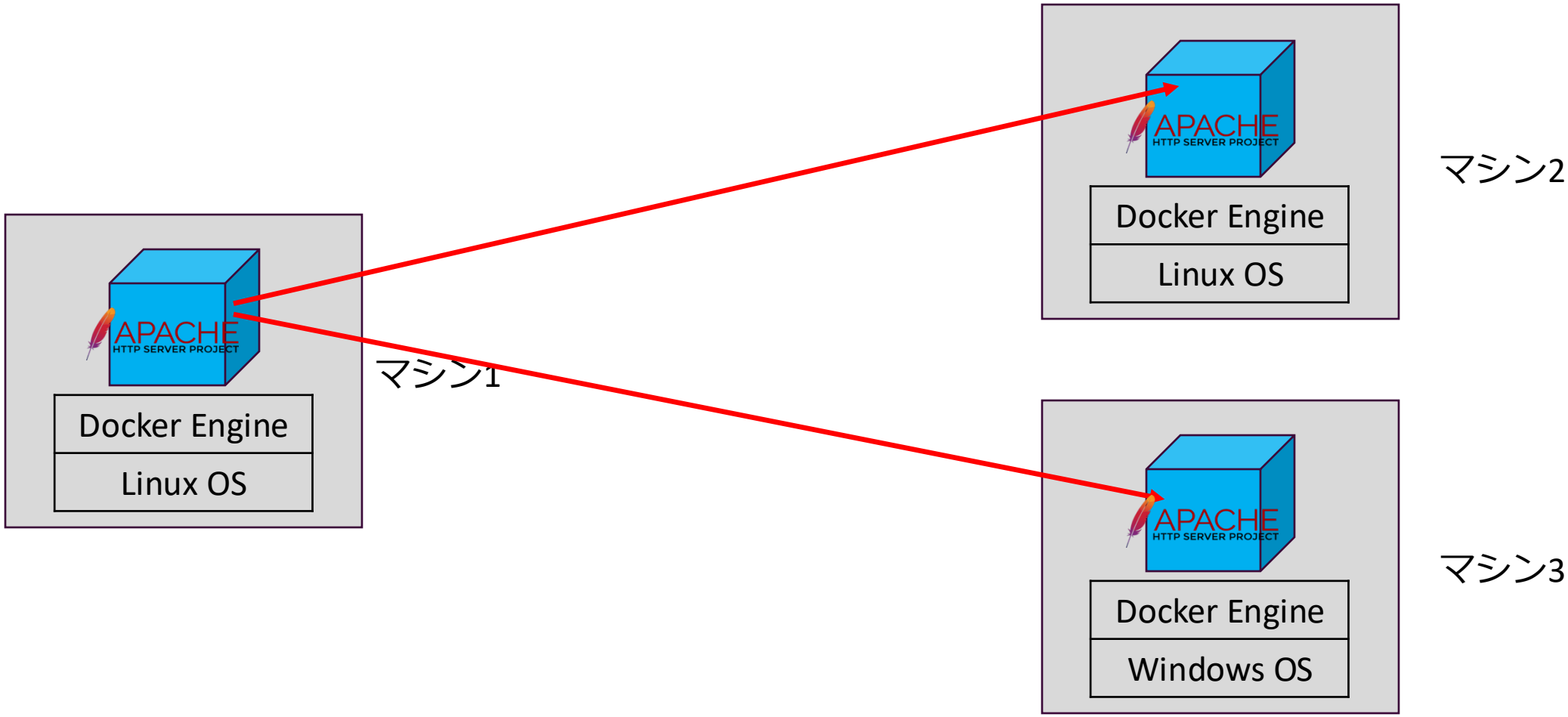
システムとライブラリをセットで隔離することで、この問題を回避することができる

メリット2: どこでも同じように動く

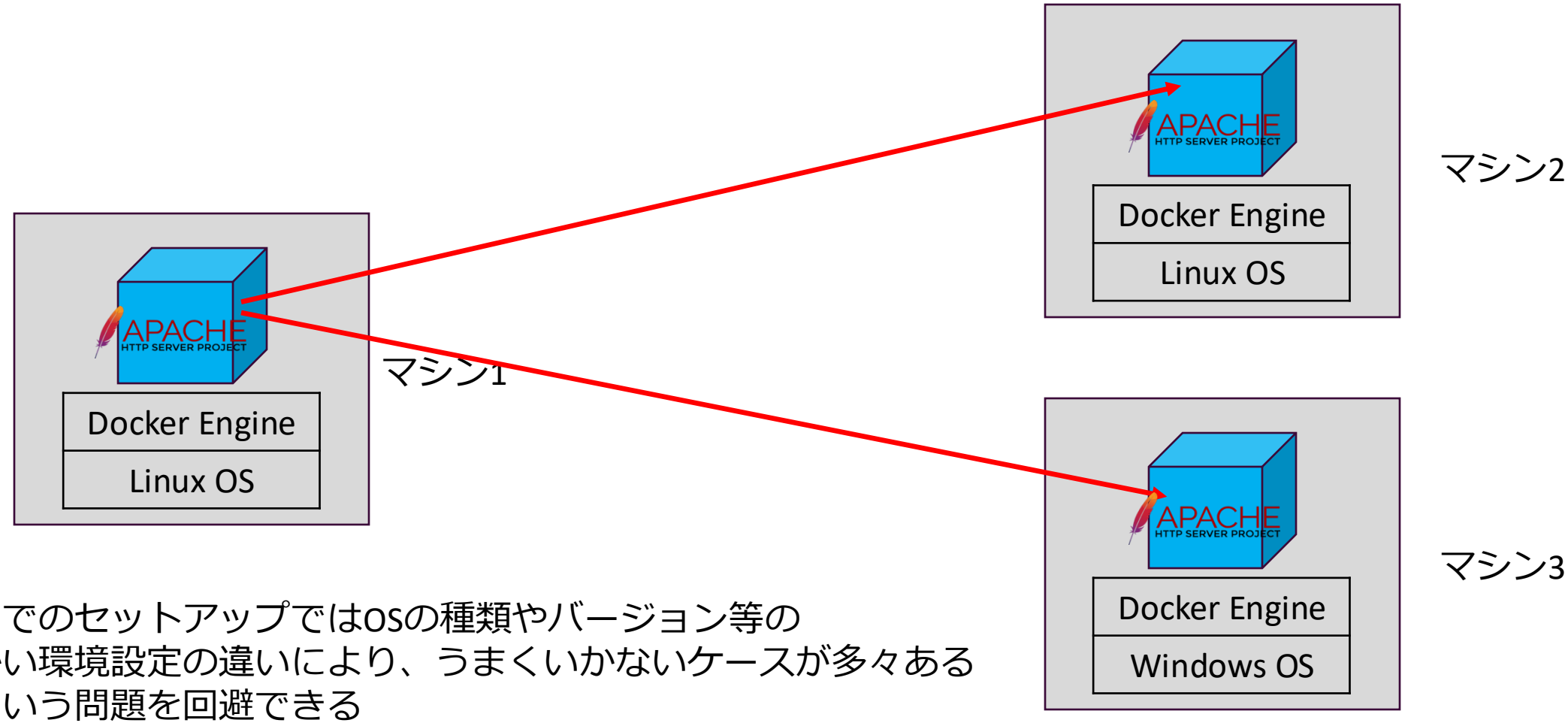


マシン1

メリット2: どこでも同じように動く



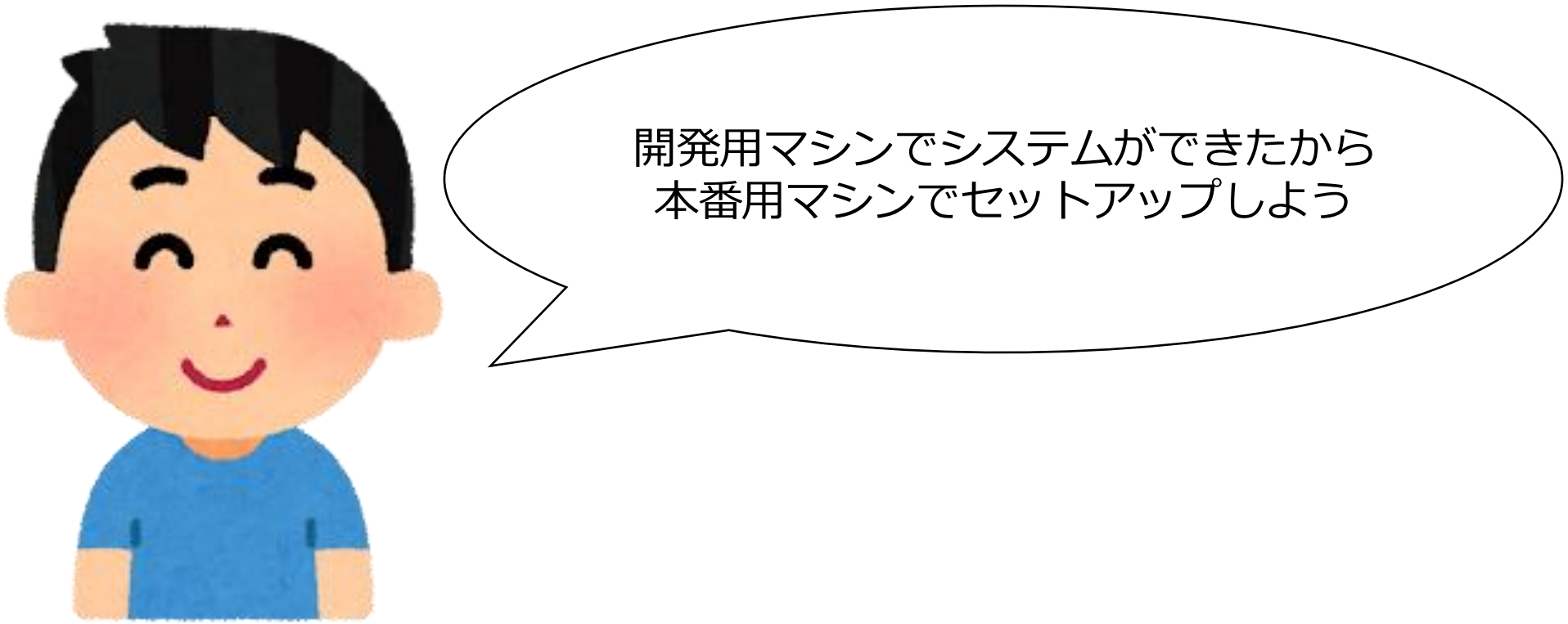
メリット2: どこでも同じように動く



手動でのセットアップではosの種類やバージョン等の
細かい環境設定の違いにより、うまくいかないケースが多々ある
そういう問題を回避できる

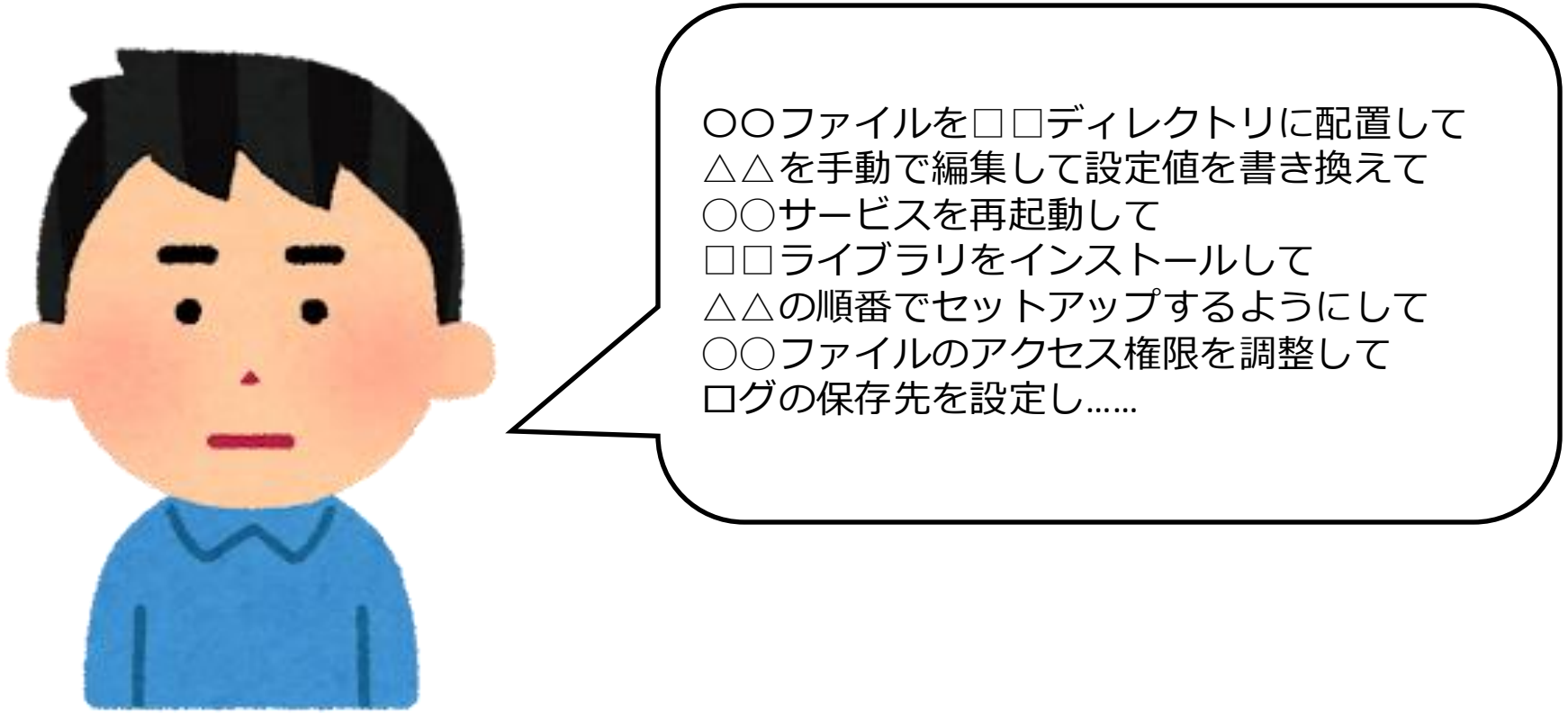
メリット3: すぐ立ち上げてすぐ削除できる

コンテナを使わずに問題となるケース (立ち上げ時)



メリット3: すぐ立ち上げてすぐ削除できる

コンテナを使わずに問題となるケース (立ち上げ時)

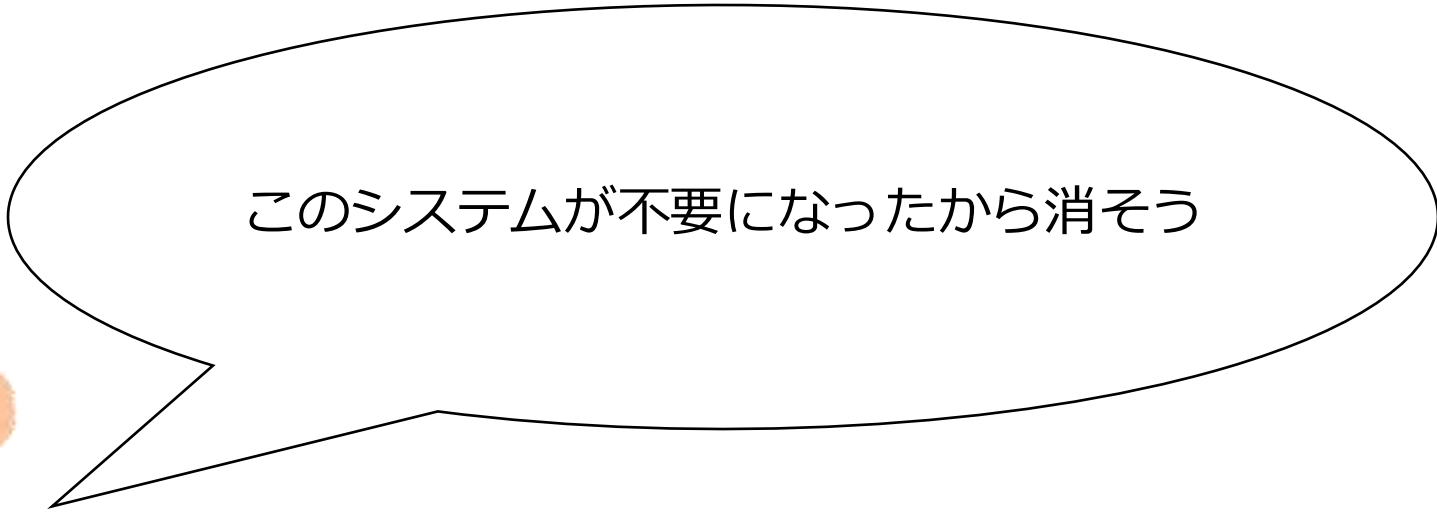


メリット3: すぐ立ち上げてすぐ削除できる

コンテナを使わずに問題となるケース (立ち上げ時)

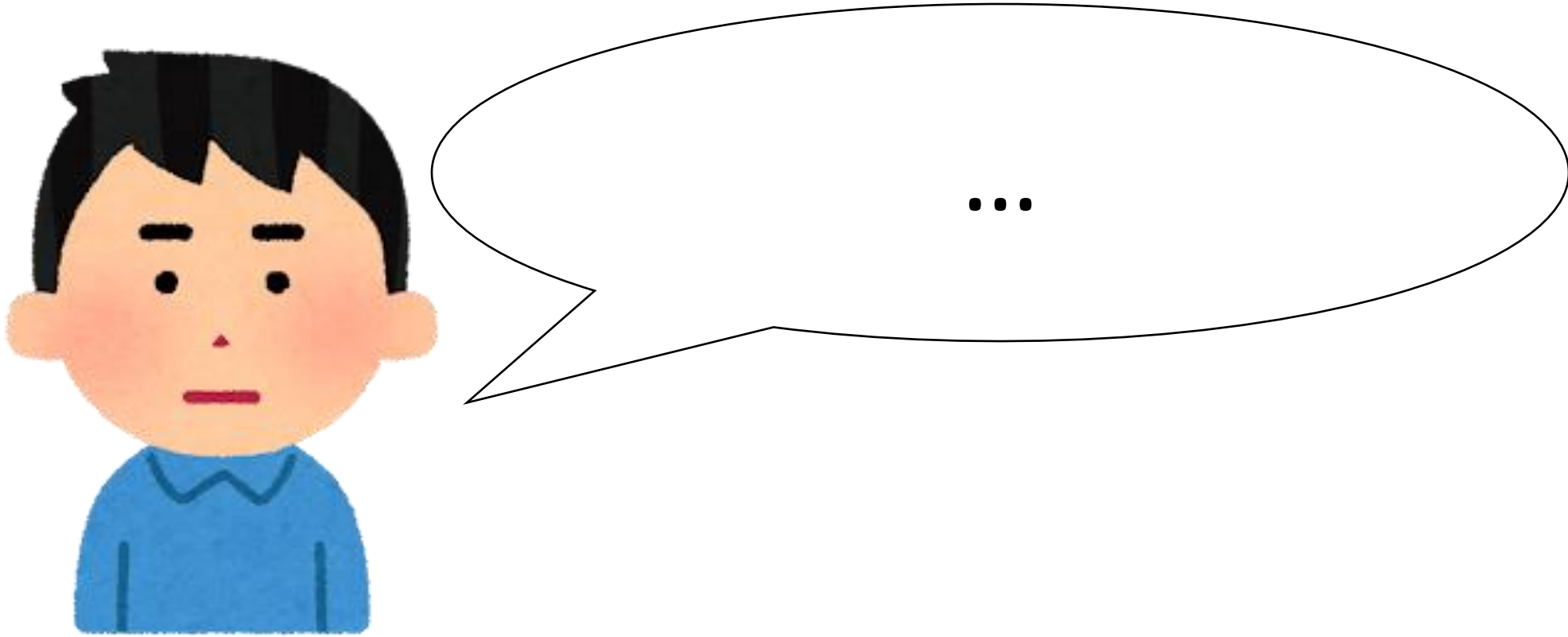
メリット3: すぐ立ち上げてすぐ削除できる

コンテナを使わずに問題となるケース (削除, クリーンアップ時)



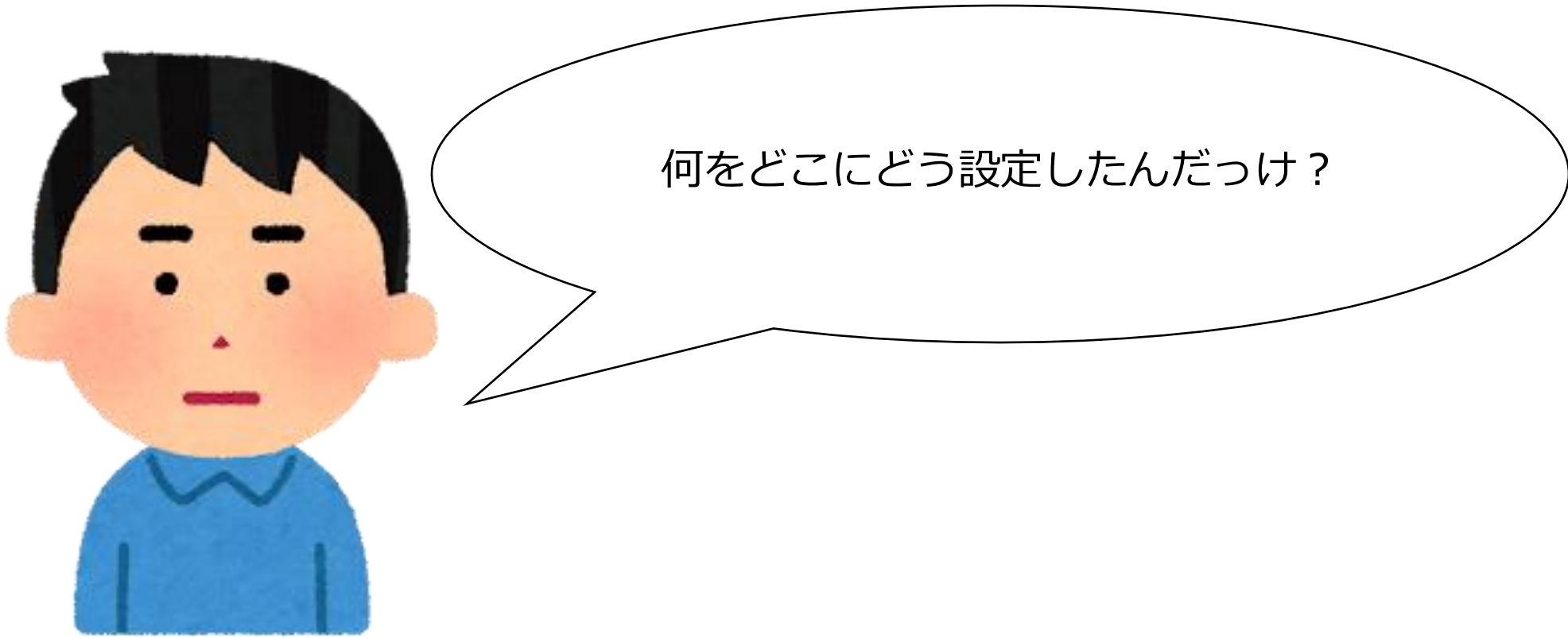
メリット3: すぐ立ち上げてすぐ削除できる

コンテナを使わずに問題となるケース (削除, クリーンアップ時)

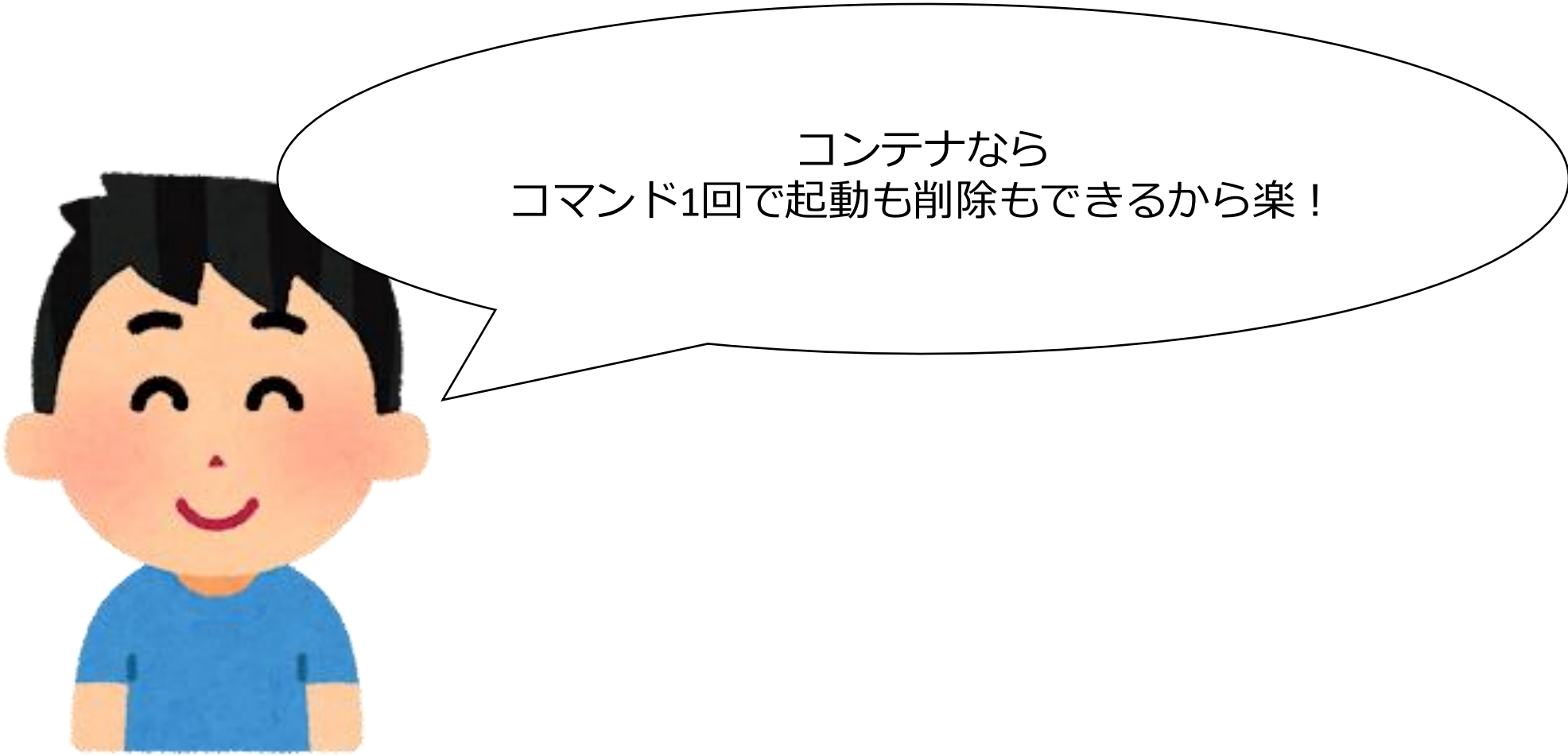


メリット3: すぐ立ち上げてすぐ削除できる

コンテナを使わずに問題となるケース (削除, クリーンアップ時)



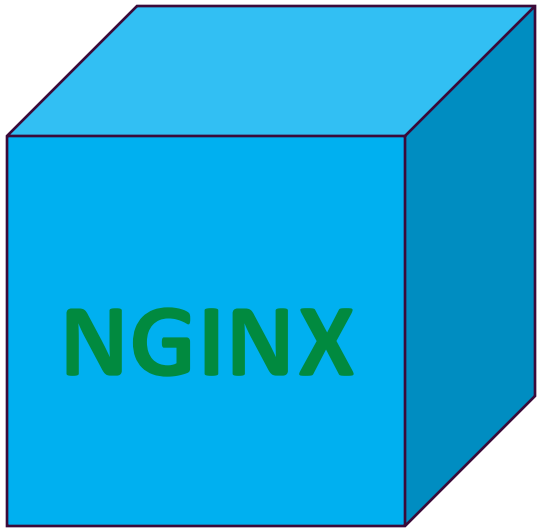
メリット3: すぐ立ち上げてすぐ削除できる



実践: サーバを立てる

1. Nginx
2. Apache
3. 手元でイメージ作成・コンテナ実行
4. 3つのコンテナを一気に実行

Dockerを使ってNginxというサーバのコンテナを立てる



Dockerでサーバを立てる: nginx

docker run -d -p 8080:80 --name my-nginx nginx

```
aic-student@team1:~$ docker run -d -p 8080:80 --name my-nginx nginx
Unable to find image 'nginx:latest' locally
latest: Pulling from library/nginx
59e22667830b: Pull complete
140da4f89dcb: Pull complete
96e47e70491e: Pull complete
2ef442a3816e: Pull complete
4b1e45a9989f: Pull complete
1d9f51194194: Pull complete
f30ffbee4c54: Pull complete
Digest: sha256:84ec966e61a8c7846f509da7eb081c55c1d56817448728924a87ab32f12a72fb
Status: Downloaded newer image for nginx:latest
c1d2fe8a615df577d6bb8422f1a4573e70f2a3dad7eec2c7e7448135ffb9023b
```

この1コマンドでnginxサーバのコンテナが立つ
だけど少し注意が必要

Dockerでサーバを立てる: nginx

docker run -d -p 8080:80 --name my-nginx nginx

docker run	イメージを実行するコマンド (＝docker container run)
-d	バックグラウンド実行 (なしで実行するとNginxのログが出力され続ける)
-p 8080:80	マシンの8080番ポートをコンテナの80番ポートに接続する
--name my-nginx	コンテナに”my-nginx”という名前をつけて管理する
nginx	使用するイメージ名

同じチームで皆んながこれを実行すると、一人以外失敗するはず

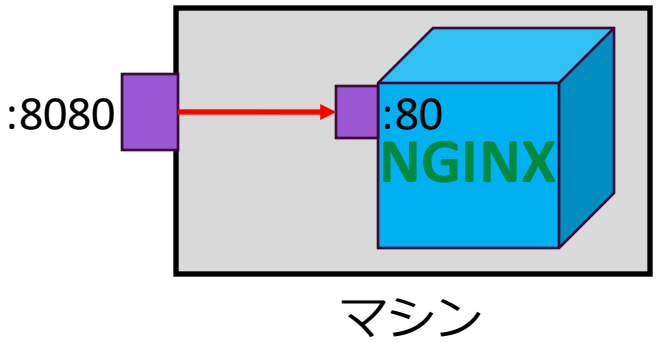
理由1: マシンの8080ポートを使えるのは1つのプロセスのみ
理由2: 同じ名前のコンテナは複数個作れない

ポート番号とコンテナ名を変更すれば皆んな実行できるはず
ポート番号: 0 ～ 65535

Dockerでサーバを立てる: nginx

docker run -d -p 8080:80 --name my-nginx nginx

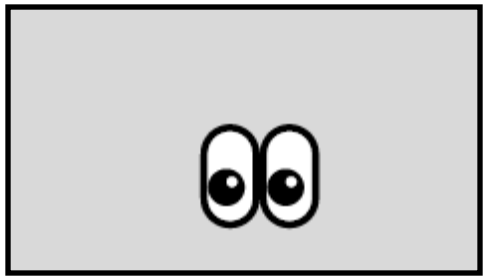
docker run	イメージを実行するコマンド (＝docker container run)
-d	バックグラウンド実行 (なしで実行するとNginxのログが出力され続ける)
-p 8080:80	マシンの8080番ポートをコンテナの80番ポートに接続する
--name my-nginx	コンテナに”my-nginx”という名前をつけて管理する
nginx	使用するイメージ名



Dockerでサーバを立てる: nginx

```
docker run -d -p 8080:80 --name my-nginx nginx
```

docker run	イメージを実行するコマンド (＝docker container run)
-d	バックグラウンド実行 (なしで実行するとNginxのログが出力され続ける)
-p 8080:80	マシンの8080番ポートをコンテナの80番ポートに接続する
--name my-nginx	コンテナに”my-nginx”という名前をつけて管理する
nginx	使用するイメージ名



マシン

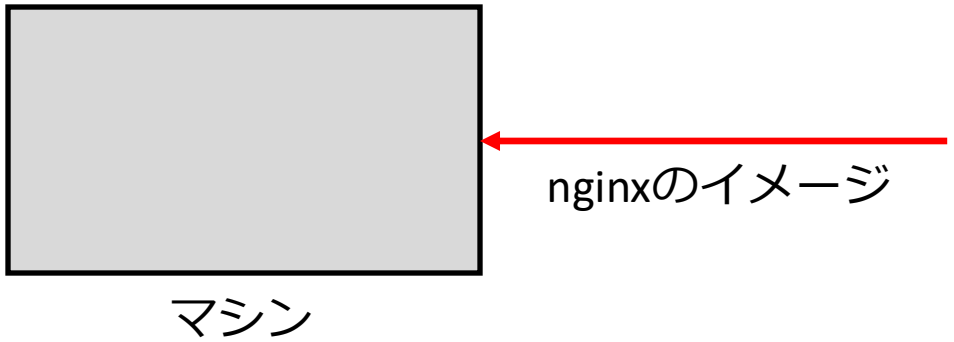
Docker Hub

まずローカルに”nginx”というイメージがあるか確認

Dockerでサーバを立てる: nginx

docker run -d -p 8080:80 --name my-nginx nginx

docker run	イメージを実行するコマンド (＝docker container run)
-d	バックグラウンド実行 (なしで実行するとNginxのログが出力され続ける)
-p 8080:80	マシンの8080番ポートをコンテナの80番ポートに接続する
--name my-nginx	コンテナに”my-nginx”という名前をつけて管理する
nginx	使用するイメージ名



様々なDockerイメージが公開されている場所

無ければ ネットワークからイメージを取得

Dockerでサーバを立てる: nginx

docker ps

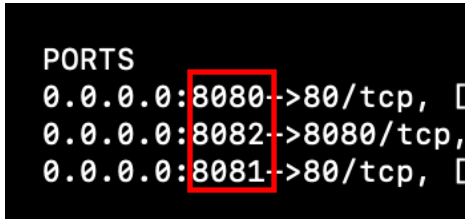
```
aic-student@team1:~$ docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED
c1d2fe8a615d   nginx     "/docker-entrypoint.…"   About a minute ago

 STATUS              PORTS                                   NAMES
 Up About a minute   0.0.0.0:8080->80/tcp, [::]:8080->80/tcp   my-nginx
```

動作しているDockerコンテナー覧を表示するコマンド
-a オプション で停止中のコンテナー覧も閲覧可能

Dockerでサーバを立てる: nginx

curl localhost:8080

docker ps の実行結果



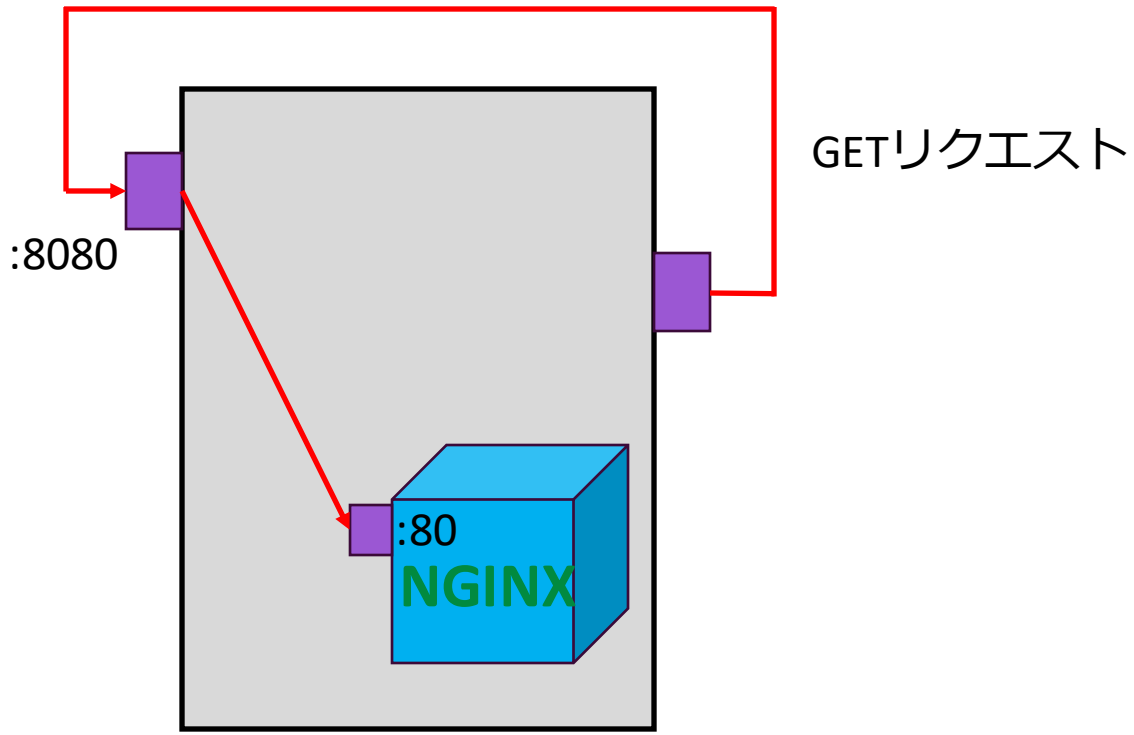
自分が実行したコンテナ名の
行を見て、
8080を置き換える

```
aic-student@team1:~$ curl localhost:8080
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

Dockerでサーバを立てる: nginx

curl localhost:8080

curl	指定したURLやアドレスにHTTPリクエストを送るコマンド
localhost:8080	localhost(そのマシン自体)の8080番ポートを指定



Dockerでサーバを立てる: nginx

docker stop my-nginx

```
aic-student@team1:~$ docker stop my-nginx
my-nginx
aic-student@team1:~$ docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
```

動作中のコンテナを終了 (Exit) させるコマンド

Dockerでサーバを立てる: nginx

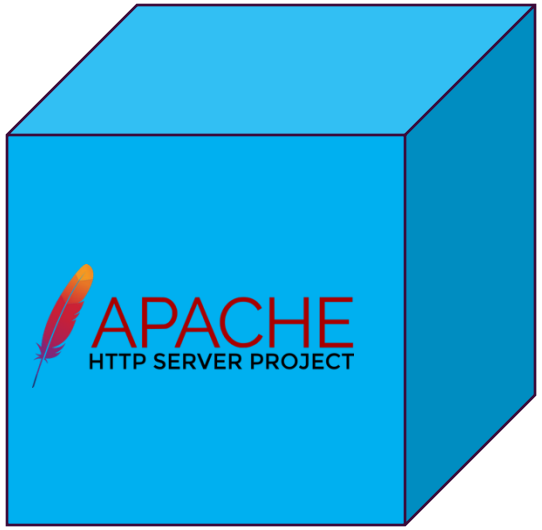
docker rm my-nginx

```
aic-student@team1:~$ docker ps -a
CONTAINER ID   IMAGE     COMMAND                  CREATED         STATUS                      PORTS     NAMES
c1d2fe8a615d   nginx     "/docker-entrypoint.…"   7 minutes ago   Exited (0) 2 minutes ago              my-nginx
aic-student@team1:~$ docker rm my-nginx
my-nginx
aic-student@team1:~$ docker ps -a
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
```

コンテナを削除するコマンド

Dockerでサーバを立てる: Apache

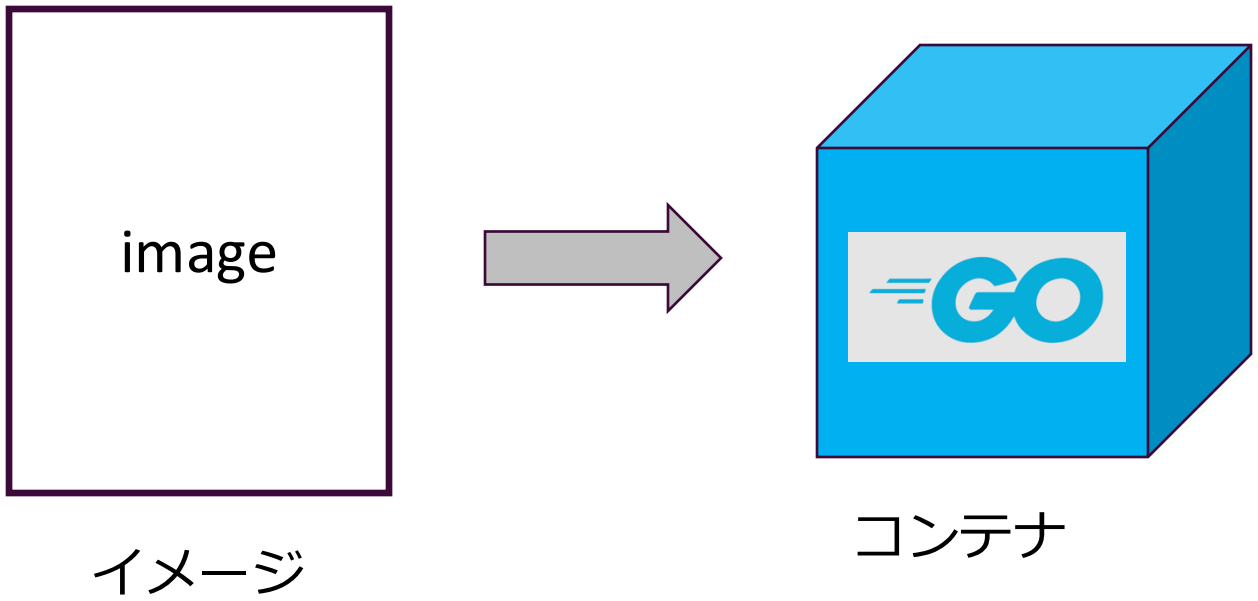
Dockerを使ってApacheというサーバのコンテナを立てる(自力で)



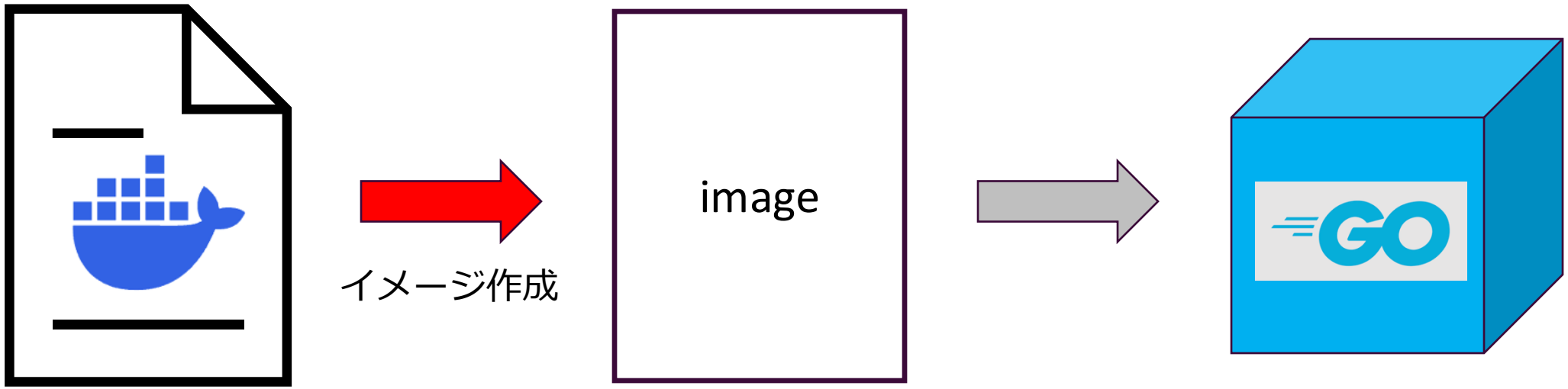
Apacheのイメージ名: httpd
ポートは8080以外がおすすめ？ 8081など

手元でイメージを作成してコンテナとして実行する

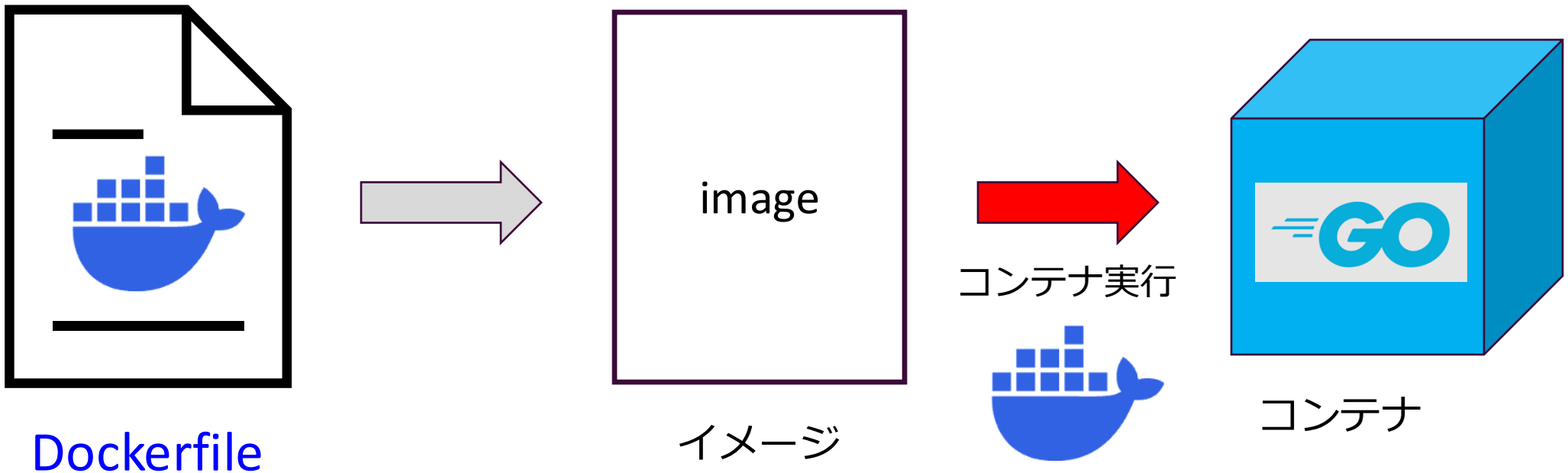
手元でイメージを作成してコンテナとして実行する



手元でイメージを作成してコンテナとして実行する



手元でイメージを作成してコンテナとして実行する



Dockerでサーバを立てる: 手元でイメージを作成

git clone https://github.com/numa123/linux_root_lecture_simple_go_app.git

```
aic-student@team1:~$ git clone https://github.com/numa123/linux_root_lecture_simple_go_app.git
Cloning into 'linux_root_lecture_simple_go_app'...
remote: Enumerating objects: 6, done.
remote: Counting objects: 100% (6/6), done.
remote: Compressing objects: 100% (5/5), done.
remote: Total 6 (delta 0), reused 6 (delta 0), pack-reused 0 (from 0)
Receiving objects: 100% (6/6), done.
```

今回使用するファイル群をダウンロード

Dockerでサーバを立てる: 手元でイメージを作成

docker build -t simple-go-app .

```
aic-student@team1:~/linux_root_lecture_simple_go_app$ docker build -t simple-go-app .
[+] Building 29.5s (9/9) FINISHED                                                                   docker:default
 => [internal] load build definition from Dockerfile                                                           0.0s
 => => transferring dockerfile: 787B                                                                           0.0s
 => [internal] load metadata for docker.io/library/golang:1.21-alpine                                          2.7s
 => [internal] load .dockerignore                                                                              0.0s
 => => transferring context: 2B                                                                                0.0s
 => [1/4] FROM docker.io/library/golang:1.21-alpine@sha256:2414035b086e3c42b99654c8b26e6f5b1b1598080d65fd03c7f499552ff4dc94  10.7s
 => => resolve docker.io/library/golang:1.21-alpine@sha256:2414035b086e3c42b99654c8b26e6f5b1b1598080d65fd03c7f499552ff4dc94  0.0s
 => => sha256:8ee9b9e11ef79e314a7584040451a6df8e72a66712e741bf75951e05e587404e 1.92kB / 1.92kB                 0.0s
 => => sha256:c2321c7cf7210be837249dba0f3699fad6ddbb5718e70344b28c0d58feff4c0b 2.10kB / 2.10kB                 0.0s
 => => sha256:c6a83fedfae6ed8a4f5f7cbb6a7b6f1c1ec3d86fea8cb9e5ba2e5e6673fde9f6 3.62MB / 3.62MB                 0.6s
 => => sha256:41db7493d1c6f3f26428d119962e3862c14a9e20bb0b8fefc36e7282d015d099 290.89kB / 290.89kB             0.7s
 => => sha256:54bf7053e2d96c2c7f4637ad7580bd64345b3c9fabb163e1fdb8894aea8a9af0 67.01MB / 67.01MB               3.3s
 => => sha256:2414035b086e3c42b99654c8b26e6f5b1b1598080d65fd03c7f499552ff4dc94 10.30kB / 10.30kB               0.0s
 => => extracting sha256:c6a83fedfae6ed8a4f5f7cbb6a7b6f1c1ec3d86fea8cb9e5ba2e5e6673fde9f6                      0.1s
 => => sha256:4579008f8500d429ec007d09232919100971 1942d9380d060c8d9bd24c0c352c 126B / 126B                    1.1s
 => => sha256:4f4fb700ef54461cfa02571ae0db9a0dc1e0cdb5577484a6d75e68dc38e8acc1 32B / 32B                       1.1s
 => => extracting sha256:41db7493d1c6f3f26428d119962e3862c14a9e20bb0b8fefc36e7282d015d099                      0.1s
 => => extracting sha256:54bf7053e2d96c2c7f4637ad7580bd64345b3c9fabb163e1fdb8894aea8a9af0                      6.7s
 => => extracting sha256:4579008f8500d429ec007d092329191009711942d9380d060c8d9bd24c0c352c                      0.0s
 => => extracting sha256:4f4fb700ef54461cfa02571ae0db9a0dc1e0cdb5577484a6d75e68dc38e8acc1                      0.0s
 => [internal] load build context                                                                              0.0s
 => => transferring context: 34.57kB                                                                           0.0s
 => [2/4] WORKDIR /app                                                                                         2.7s
 => [3/4] COPY . .                                                                                             0.1s
 => [4/4] RUN go build -o server .                                                                            12.4s
 => exporting to image                                                                                         0.8s
 => => exporting layers                                                                                        0.8s
 => => writing image sha256:a1e95b8c01d0b72b0dfc5dafff611ee61f189e1f80cca652e968c4d9d91f89a3                   0.0s
 => => naming to docker.io/library/simple-go-app                                                               0.0s
```

cd linux_root_lecture_simple_go_app で Dockerfileのあるディレクトリに移動してから実行

Dockerでサーバを立てる: 手元でイメージを作成

```
docker build -t simple-go-app .
```

docker build	Dockerfileを元にDockerイメージを作成するコマンド
-t simple-go-app	作るイメージを“simple-go-app”という名前にする
.	現在のディレクトリにある Dockerfile を参照する

これも同じコマンドを実行しようとすると一人以外失敗するはず

理由: 同じ名前で複数のイメージ名は作成できない

名前を各自変えれば実行できるはず

Dockerでサーバを立てる: 手元でイメージを作成

docker image ls

```
aic-student@team1:~/linux_root_lecture_simple_go_app$ docker image ls
REPOSITORY      TAG       IMAGE ID       CREATED          SIZE
simple-go-app   latest    a1e95b8c01d0   2 minutes ago    292MB
httpd           latest    5bdbc621ec08   13 days ago      148MB
nginx           latest    2cd1d97f893f   3 weeks ago      192MB
```

ローカルに存在する Docker イメージの一覧を表示する

Dockerでサーバを立てる: 手元でイメージを作成

docker run -d -p 8082:8080 --name my-simple-go-app simple-go-app

curl localhost:8082

```
aic-student@team1:~/linux_root_lecture_simple_go_app$ docker run -d -p 8082:8080 --name my-simple-go-app simple-go-app
920afdf3a31b79def7da258cb1c81a3817d438cd35c197a33afb2dcad9ac55e8
aic-student@team1:~/linux_root_lecture_simple_go_app$ curl localhost:8082
Hello
```

```go
func handler(w http.ResponseWriter, r *http.Request) {
    fmt.Fprintln(w, "Hello")
}
```

Helloと返ってくるはず

docker ps でも動いていることを確認

Dockerでサーバを立てる: 手元でイメージを作成

Dockerfileの説明

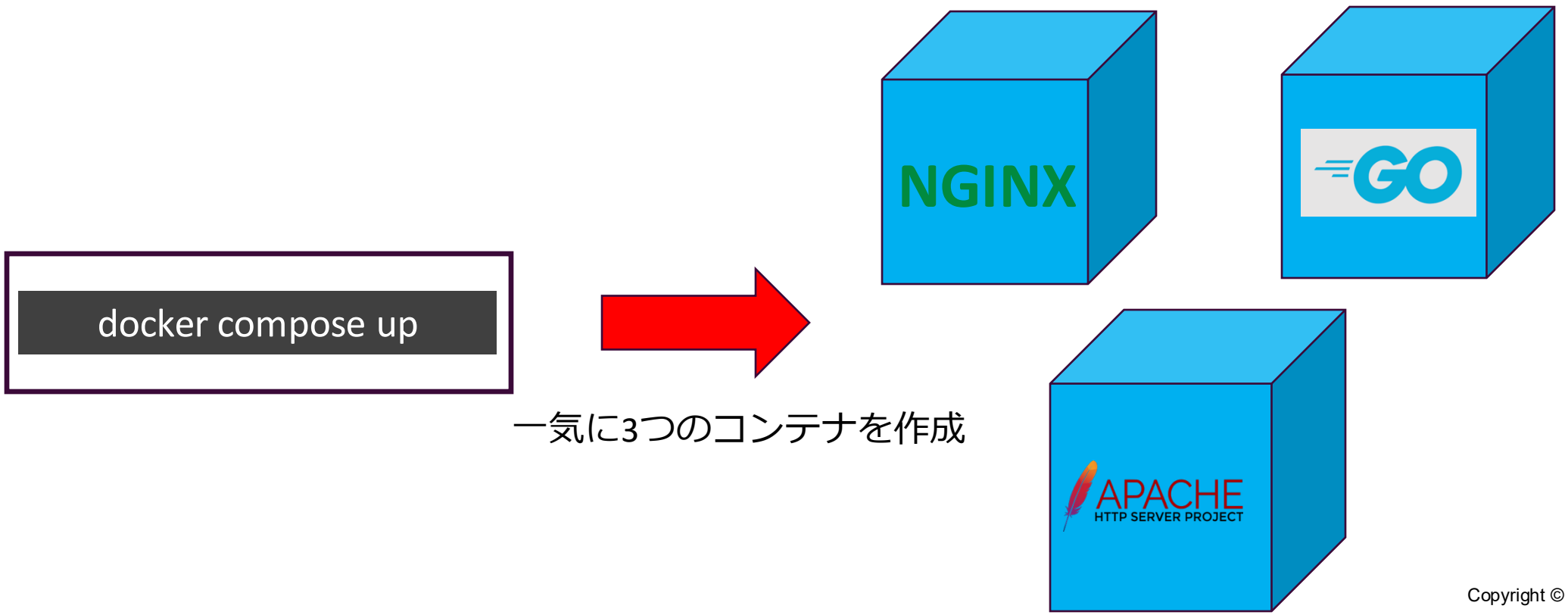
```
1   # Go言語とAlpine Linux（超軽量なLinux）の入ったベースイメージを使用
2   FROM golang:1.24-alpine
3
4   # 作業ディレクトリを /app に設定（ここにファイルを置いて作業する）
5   WORKDIR /app
6
7   # 現在のディレクトリ（ソースコードなど）をコンテナの /app にコピー
8   COPY . .
9
10  # Goのビルドコマンド。main.go などをビルドして "server" という実行ファイルを作成
11  RUN go build -o server .
12
13  # コンテナ起動時に実行されるコマンド（ビルドした "server" を実行）
14  CMD ["./server"]
15
```

Docker Composeを使って一気に3つのサーバ(コンテナ)を立てる

Docker Compose: 1つ以上のコンテナをまとめて管理できる機能

Docker Composeを使って一気に3つのサーバ(コンテナ)を立てる

Docker Compose: 1つ以上のコンテナをまとめて管理できる機能



Dockerでサーバを立てる: Docker Composeを使う

compose.yaml のあるディレクトリに移動

```
aic-student@team1:~/linux_root_lecture_simple_go_app$ ls
Dockerfile  compose.yaml  go.mod  main.go
```

Dockerでサーバを立てる: Docker Composeを使う

docker compose up -d

```
aic-student@team1:~/linux_root_lecture_simple_go_app$ docker compose up -d
#1 [internal] load local bake definitions
#1 reading from stdin 631B done
#1 DONE 0.0s

#2 [internal] load build definition from Dockerfile
#2 transferring dockerfile: 787B done
#2 DONE 0.0s

#3 [internal] load metadata for docker.io/library/golang:1.21-alpine
#3 DONE 1.3s

#4 [internal] load .dockerignore
#4 transferring context: 2B done
#4 DONE 0.0s

#5 [1/4] FROM docker.io/library/golang:1.21-alpine@sha256:2414035b086e3c42b99654c8b26e6f5b1b1598080d65fd03c7f499552ff4dc94
#5 DONE 0.0s

#6 [internal] load build context
#6 transferring context: 2.85kB done
#6 DONE 0.0s

#7 [2/4] WORKDIR /app
#7 CACHED

#8 [3/4] COPY . .
#8 DONE 0.1s

#9 [4/4] RUN go build -o server .
#9 DONE 12.6s

#10 exporting to image
#10 exporting layers
#10 exporting layers 0.8s done
#10 writing image sha256:b4e54646b9a62bf56529f330482edac207d883f778c2e5e6b47df1b79cc2a00f done
#10 naming to docker.io/library/linux_root_lecture_simple_go_app-simple-go-app done
#10 DONE 0.8s

#11 resolving provenance for metadata file
#11 DONE 0.0s
[+] Running 5/5
 ✔ linux_root_lecture_simple_go_app-simple-go-app    Built                                0.0s
 ✔ Network linux_root_lecture_simple_go_app_default  Created                              0.1s
 ✔ Container simple-go-server                        Started                              0.4s
 ✔ Container apache-server                           Started                              0.4s
 ✔ Container nginx-server                            Started                              0.3s
```

Dockerでサーバを立てる: Docker Composeを使う

docker compose up -d

docker compose up	compose.yamlに書かれたサービスを起動してコンテナを作成・実行する
-d	バックグラウンド実行

このポートも被らないように
書き換えてみて

```yaml
services:  # コンテナの設定
  Run Service
  nginx:  # nginxコンテナ
    image: nginx:latest  # nginxイメージを使用
    container_name: nginx-server  # コンテナ名
    ports:
      - "8083:80"  # ポートマッピング。ホスト:コンテナ

  Run Service
  apache:  # apacheコンテナ
    image: httpd:latest  # apacheイメージを使用
    container_name: apache-server  # コンテナ名
    ports:
      - "8084:80"  # ポートマッピング。ホスト:コンテナ

  Run Service
  simple-go-app:  # Goアプリケーション
    build: .  # カレントディレクトリのDockerfileを使用
    container_name: simple-go-server  # コンテナ名
    ports:
      - "8085:8080"  # ポートマッピング。ホスト:コンテナ

```

Dockerでサーバを立てる: Docker Composeを使う

docker ps で3つのコンテナが立っているか確認

```
aic-student@team1:~/linux_root_lecture_simple_go_app$ docker ps
CONTAINER ID   IMAGE                                      COMMAND
a9e703a84f35   nginx:latest                               "/docker-entrypoint.…"
abc7171af6ba   httpd:latest                               "httpd-foreground"
38cad683cbd8   linux_root_lecture_simple_go_app-simple-go-app   "./server"
```

```
CREATED              STATUS              PORTS                                       NAMES
About a minute ago   Up About a minute   0.0.0.0:8083->80/tcp, [::]:8083->80/tcp     nginx-server
About a minute ago   Up About a minute   0.0.0.0:8084->80/tcp, [::]:8084->80/tcp     apache-server
About a minute ago   Up About a minute   0.0.0.0:8085->8080/tcp, [::]:8085->8080/tcp   simple-go-server
```

Dockerでサーバを立てる: Docker Composeを使う

curl で確認

```
[aic-student@team1:~/linux_root_lecture_simple_go_app$ curl localhost:8083
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
[aic-student@team1:~/linux_root_lecture_simple_go_app$ curl localhost:8084
<html><body><h1>It works!</h1></body></html>
[aic-student@team1:~/linux_root_lecture_simple_go_app$ curl localhost:8085
Hello
```

Dockerでサーバを立てる: Docker Composeを使う

docker compose down

```
[aic-student@team1:~/linux_root_lecture_simple_go_app$ docker compose down
 [+] Running 4/4
  ✔ Container nginx-server                                  Removed
  ✔ Container apache-server                                 Removed
  ✔ Container simple-go-server                              Removed
  ✔ Network linux_root_lecture_simple_go_app_default        Removed
```

docker compose up で作成したコンテナを全て停止・削除するコマンド

Dockerでサーバを立てる: Docker Composeを使う

compose.yaml の説明

```yaml
services:  # コンテナの設定
  nginx:  # nginxコンテナ
    image: nginx:latest  # nginxイメージを使用
    container_name: nginx-server  # コンテナ名
    ports:
      - "8083:80"  # ポートマッピング。ホスト:コンテナ

  apache:  # apacheコンテナ
    image: httpd:latest  # apacheイメージを使用
    container_name: apache-server  # コンテナ名
    ports:
      - "8084:80"  # ポートマッピング。ホスト:コンテナ

  simple-go-app:  # Goアプリケーション
    build: .  # カレントディレクトリのDockerfileを使用
    container_name: simple-go-server  # コンテナ名
    ports:
      - "8085:8080"  # ポートマッピング。ホスト:コンテナ
```

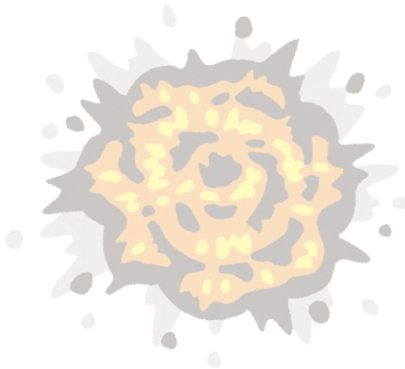
本日の流れ

概論
「大学のサバ管」

Linux管理者のための
基本操作

Webサーバ
on Docker

トラブルシューティング



破壊

トラブルシューティング回

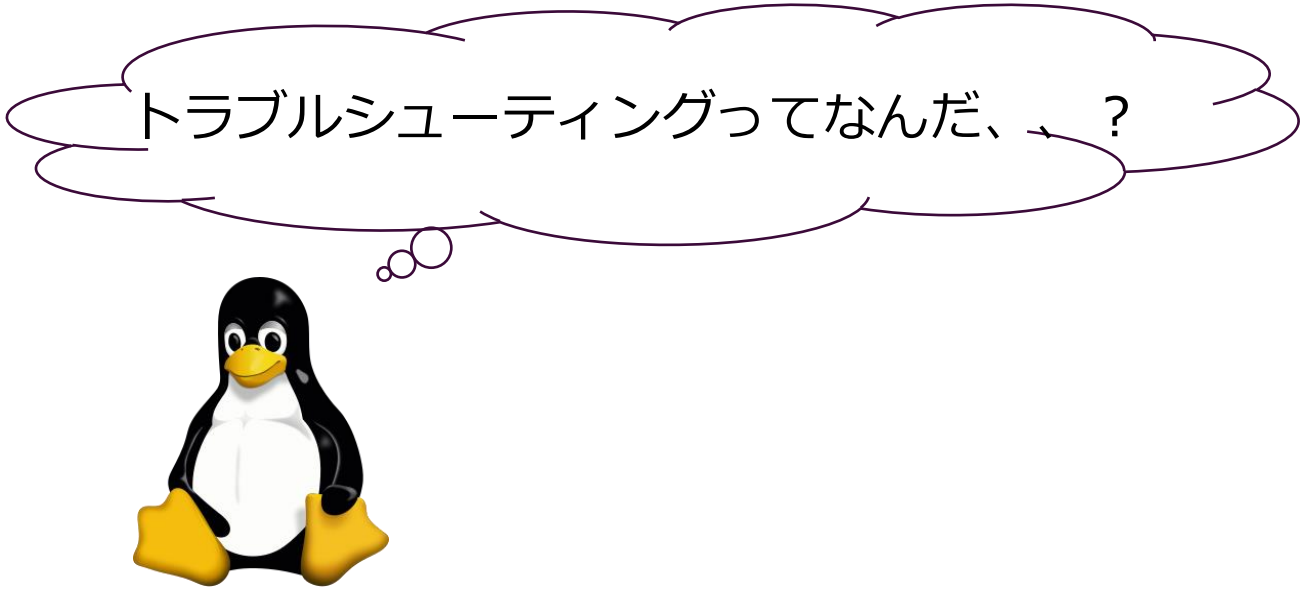
AICサーバ班 メンバー

慶應義塾大学 理工学部 情報工学科 3年 岩﨑一樹

今回のゴール

- **トラブルシューティングの手順**を身につける！
- 実務でよくある**典型トラブル**や**脆弱性の修正**を体験する！
- トラブルシューティングを**少しだけでも楽しい**と思えるようにする、、！

トラブルシューティングとは？



発生した問題の**原因を特定**し、**解決策を見つけ出す**こと！

特に、Web ページなどユーザーがよく利用するサービスにおいて問題が発生したときは**迅速な対応**をすることが重要！！

ICT トラブルシューティングコンテスト (ICTSC)

トラブルシューティングの正確性・スピードを競うコンテスト

ICTSC
ICT TroubleShooting Contest

今年（2025 年）の予定

8/30	一次予選（選択問題形式、オンライン）
12/13	二次予選（実技問題、オンライン）
3/14,15	本選（実技問題、オフライン）

問題例

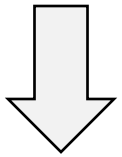
- 「**Web サーバを構築**して！」
- 「ネットワークがつながらないから、**ネットワーク機器の設定**をして！」
- 「サーバーが乗っ取られたかもしれないので、**異変がないか**調べて！」

対応の流れ

1．実際に環境を操作し、問題の**原因を特定**
2．**修正**方法を検討
3．解決策を**報告書**として提出

問題例

- 「**Web サーバを構築**して！」
- 「ネットワークがつながらないから、**ネットワーク機器の設定**をして！」
- 「サーバーが乗っ取られたかもしれないので、**異変がないか**調べて！」



- トラブルシューティングに加え、サービスの構築やセキュリティ問題の調査・解決など**幅広く出題**される。
- データベースやリバースプロキシを含む各種サービス、ネットワーク、セキュリティに関する**総合的な知識**が求められる！

トラブルシューティングのモチベ

何かを実現するときに**トラブル**はつきもの、、

トラブルシューティングのモチベ

あらかじめ技術的な知識をもっていれば、、、

迅速に対応できる　＆　**みんなのヒーロー**になれる！！！

問題の紹介（公開用）

練習問題	Docker コンテナの立ち上げ　(10pt)
第１問	ポートバインディング競合　(20pt)
第２問	SSH接続不可　(20pt)
第３問	証明書不良　(25pt)
第４問	権限昇格の脆弱性　(25pt)

公開資料においては問題の概要のみを記載しており、
詳細は講習会にて取り扱います。

本日の流れ

概論
「大学のサバ管」

Linux管理者のための
基本操作

Webサーバ
on Docker

トラブルシューティング

破壊

最後に

環境を吹っ飛ばしましょう!

世界サーバ投げ選手権

https://www.youtube.com/watch?v=IFE0JlMXCdE

世界サーバ投げ選手権

世界サーバー投げ選手権2024 ()

https://gigazine.net/news/20240304-world-server-throwing-championship/

世界サーバ投げ選手権

本心は投げたいけど今回は...

https://gigazine.net/news/20240304-world-server-throwing-championship/

すべてを消すコマンド

rm –rf /

rm : ファイル、ディレクトリの消去
-r : ディレクトリ内のファイルを再帰的に消去
-f : 確認なし強制的に消去
/ : rm 対象にルートを指定

流石にrm –rf / はそのまま実行はできない

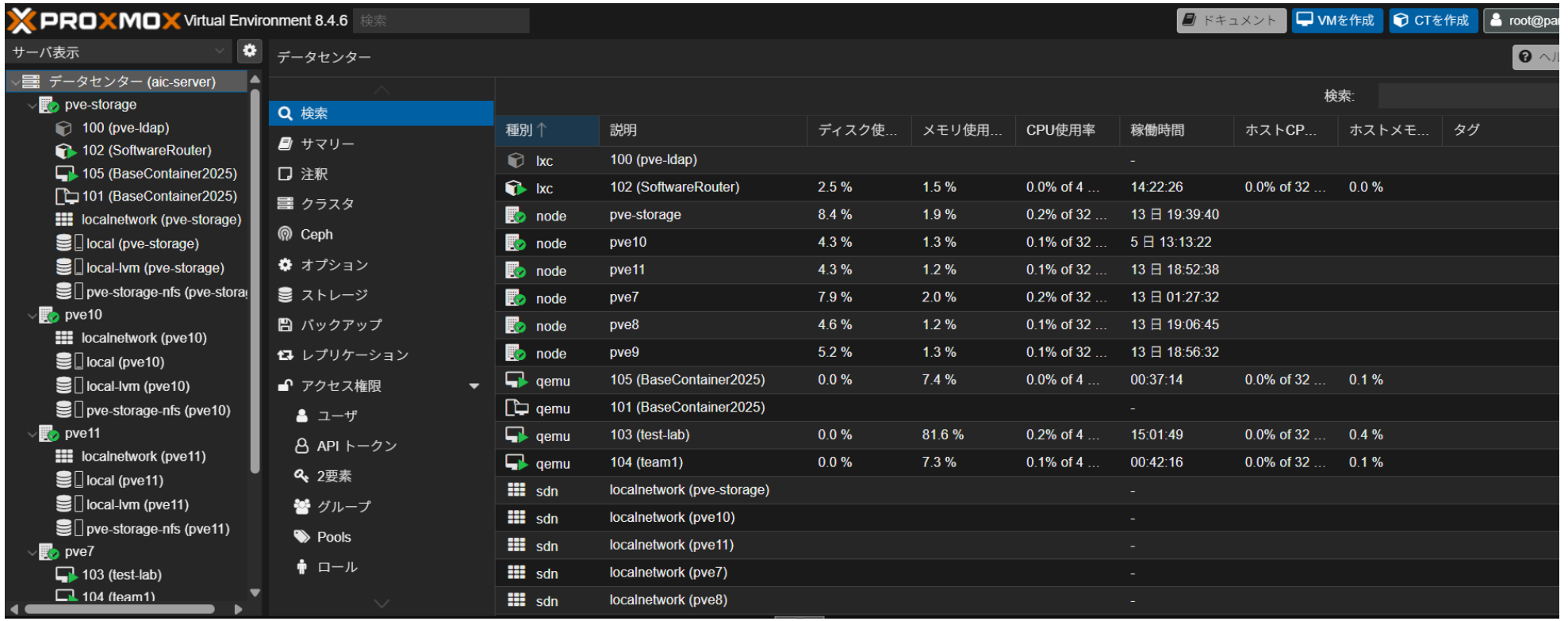
rm –rf 自体はよく使うコマンドのため、
打ち間違えによる大事故を防ぐため
rm –rf / を本当に実行するときは

rm –rf / --no-preserve-root

のオプションが必要

どうなってるかを確認しましょう

sshセッションは途切れてしまいますのでこちらで
rm –rf / の後の世界を見てみましょう



どうなってるかを確認しましょう

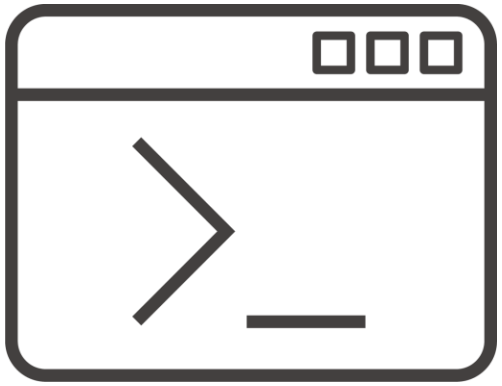
rm –rf / をしても既に開いているシェルは終了しない
… 実行しているアプリはメモリ上に存在するため

しかし、ls, catなど組み込みコマンド以外のコマンドは
すべて使えなくなっている
… コマンド実行ファイルがすべて消されたため

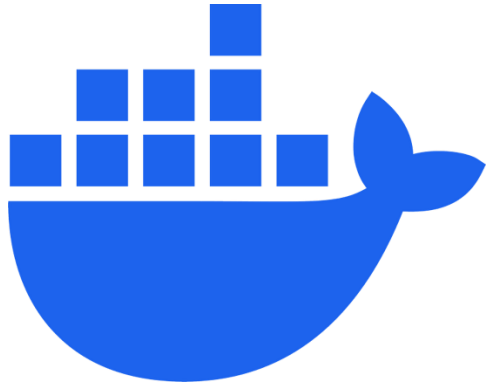
再起動ができず、強制終了後起動ができなくなる

まとめ

概論
「大学のサバ管」



Linux管理者のための
基本操作



Webサーバ
on Docker

トラブルシューティング

破壊