サーバエンジニア講座

はじめに

はじめに

- 巷に売られている本や、インターネットからは、豊富な知識を得ることができる。

- しかし、情報量が多く、初心者の壁となることも少なくない。

- そこで、サーバ管理に必要最低限な項目に絞り、効率的に学習するのが本講座の趣旨である。

演習環境について

- 本講座では、情報研究会所有のサーバに構築した演習環境に接続し、実習を行う。
- 実機を用いることにより、直感的に学習することができる。
- 学校の演習環境と違い、管理者への昇格が可能。

準備

演習環境への接続

- 演習環境を使用するためには、専用のネットワークに接続する必要がある。

- 「vpnux Client」を起動し、演習用ネットワーク接続プロファイルを選び、接続する。

資格情報

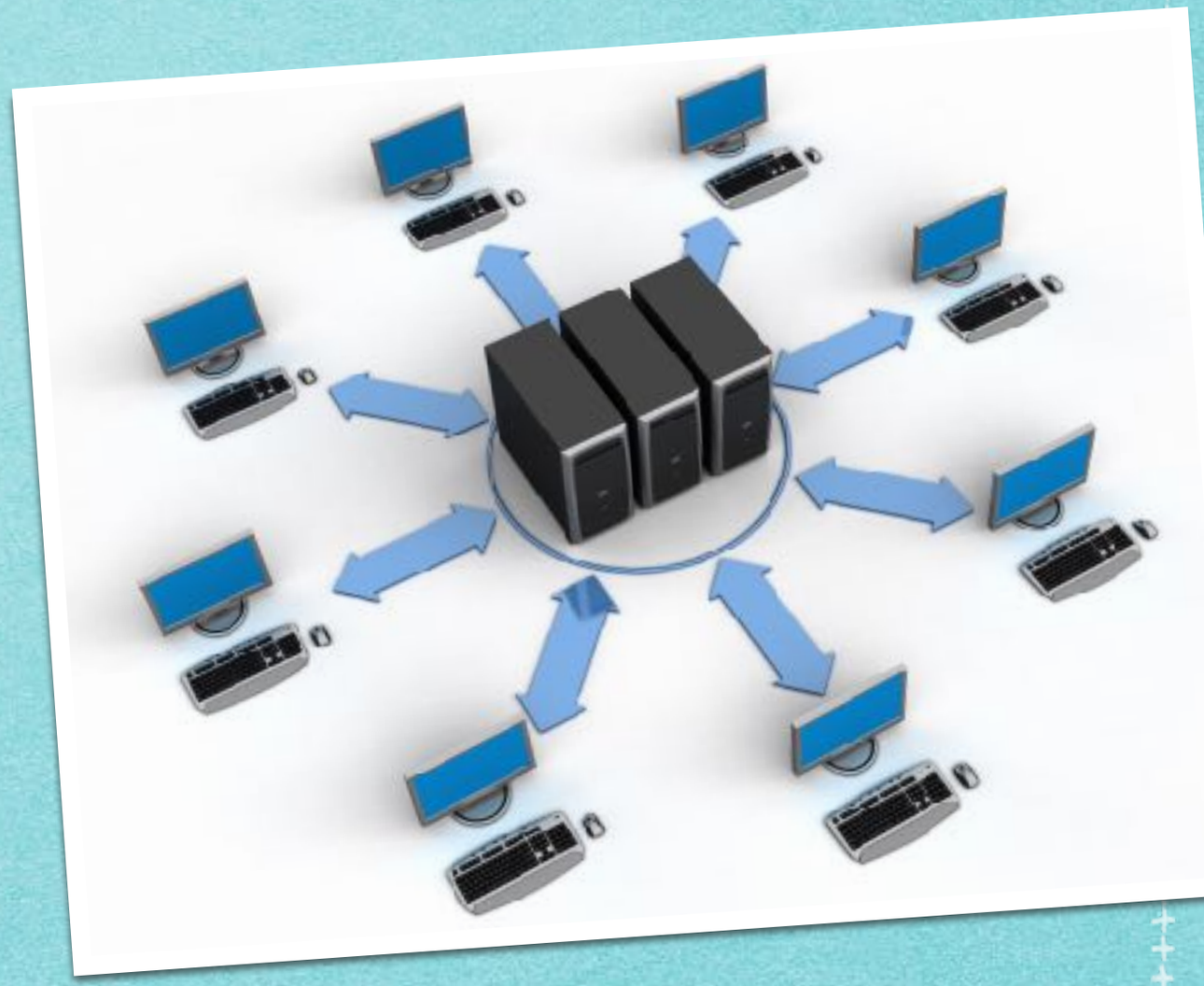
- 演習用ネットワークに接続したら、TeraTermに以下を入力し、演習環境にログインする。
  - ホスト名　：　配布した資格情報を参照
  - ユーザ名　：　vagrant
  - パスワード：　配布した資格情報を参照

予備知識

そもそもサーバとは

- 複数のコンピュータが接続されたネットワークにおいて、「何らかのサービスを提供するコンピュータ」のこと。

- Webサーバであれば、「Webページのデータを配信するサービス」を提供している。



Linux

- １９９６年、フィンランドの大学院生であるリーナス・トーバルズ氏によって開発されたOS。

- 様々な派生パッケージ（ディストリビューション）が存在する。

なぜLinux？

- 無料で提供されている（有料のものもある）。
- 非常に安定して動作する。
- パッチの提供が迅速。

サーバ用OS

- 家庭向けでは、「Windows」、「Macintosh」が一般的である。

- 同じく、サーバ用途には「Cent OS」というディストリビューションが事実上の標準である。

管理者権限

管理者権限

- コンピュータを使用するユーザは「一般ユーザ」と「管理者ユーザ」の２つに大別される。

- 管理者ユーザは、一般ユーザが許可されていない操作でも実行することができる。

- 管理者ユーザは、一般的に「root」というユーザ名である。

管理者への昇格、その前に

- $ sudo passwd root
  - 指定されたユーザのパスワードを変更する。
  - 初期状態ではrootのパスワードが未設定で、ログインできないので、パスワードを設定する。
  - sudoはおまじない。後述。

管理者権限の取得

- $ su
  - rootユーザに切り替える。
  - パスワードは表示されないが、入力は受け付けている。
  - プロンプトの$が#に変わる。以後、本資料でもこの表記に従う。

ログアウト

- # exit
  - ログアウトし、一般ユーザに戻る。
  - 一般ユーザでこのコマンドを実行すると、セッションを終了することができる。
  - 基本的に、管理者権限が必要な作業が終わったら、ログアウトしておくことが望ましい。

パッケージ管理システム

パッケージ管理システム

- システムへのパッケージ（ソフトウェア）導入、更新、削除等を行うツール。
- スマホでいう、アプリストアのような存在。
- Cent OSでは、「yum」というパッケージ管理システムが搭載されている。

パッケージ管理システム

- パッケージ管理システムでは、入手できるパッケージのリスト（カタログのようなもの）が提供されている。
- パッケージを検索、導入する時は、このリスト中から選択することになる。

パッケージリスト更新

- $ yum check-update
  - 最新のパッケージリストを取得する。
  - パッケージが更新されても、リストに載っているバージョンが古いままでは更新されないため、定期的に最新のリストを取得する必要がある。

アップデート

- # yum update
  - パッケージをアップデートする。
  - 最新のパッケージリストを取得してから実行すること。
  - 初回の実行では、処理が終わるまで長い時間が掛かるので、講習会中に実行しないこと。

試しに

- 今回は、ディレクトリ構造をツリー形式で表示するコマンド「tree」をインストールしてみよう。

検索

- $ yum search [Package]
  - パッケージを検索する。
  - $ yum search tree

インストール

- # yum install [Package]
  - パッケージをインストールする。
  - インストールは管理者権限が必要。
  - # yum install tree

依存性解決

- インストールするパッケージが、また別のパッケージを必要とする場合がある。これを「依存性」という。

- yumは、そのようなパッケージの依存性を自動解決してくれる。

依存性解決

- # yum install gcc
  - cpp、glibc-devel、kernel-headers、他数個のパッケージに依存していることがわかる。
  - さらに、cpp、glibc-develが依存しているパッケージがないか、再帰的に依存性解決を行っている。

Emacs

- 初期状態では、Emacsがインストールされていない。
- 検索するとパッケージが複数出てくる。無印のパッケージを選択しておけば、ほぼ間違いないが、この演習環境にGUIは存在しないため、GUI向けのEmacsが含まれていないであろう「emacs-nox」が最適であると考えられる。

X Window System

- 「X Window System」とは、UNIX系OSで利用されるGUI環境である。

- UNIXで「X」といったら、ほぼこれを指している。

プロセス

プロセス

- ユーザが実行するなどして、処理を行っている
  プログラムを「プロセス」と言う。

- 実行中のプロセスが、また別のプロセスを実行
  することもある（プロセスの親子関係）。

実行中のプロセスを見る

- $ pstree
  - 実行中の全プロセスを階層構造で表示する。
  - 様々なプロセスの大元は「systemd」である。
  - pstreeコマンドもプロセスの一種である。どこにあるか探してみよう。

実行中のプロセスを見る

- $ ps -ef
  - 実行中の全プロセスを表示する。

- UID　User ID
- PID　Process ID
- PPID　Parent PID
- CMD　Command

実行中のプロセスを見る

- 一般的に「grep」コマンドで目的のプロセスのみに絞り込む。

- $ ps -ef | grep [Process]

シグナル

シグナル

- UNIXにおけるシグナルとは、プロセスに送信する特定の信号のことである。

- シグナルには、HUP（ハングアップ）、INT（割込み）、KILL（強制終了）、TERM（終了）、他多数がある。

- シグナルを受信したプロセスは、そのシグナルに応じた処理を行う。

シグナルの送信

- $ kill [Signal] [PID]
  - プロセスにシグナルを送信する。
  - [Signal]を省略すると、TERMが送信される。

どんなときに使う？

- コマンドの最後に＆を付けると、バックグラウンドで実行することができる。
- $ ./endless &
- そのプロセスが暴走した場合、止める手段として、Ctrl+Cは使えない。

具体的に

- そのようなときに、KILLを送信し、プロセスを強制終了する。
- $ ps -ef | grep endless
- $ kill -KILL [PID]
- TERMで終了できない場合のみ、KILLを使う。

アーカイブと圧縮

アーカイブ

- 複数のファイルをまとめて一つのファイルにすることを「アーカイブ」と呼ぶ。

- $ tar cf files.tar file1.txt file2.txt
  - -c　アーカイブを作成。
  - -f　指定したファイル名で作成。ここでは「files.tar」。

アーカイブからの展開

- $ tar xf files.tar
  - -x　アーカイブからファイルを展開する。

圧縮・解凍

- $ gzip [File]
  - 引数で指定されたファイルを「GNU Zip」形式で圧縮する。
- $ gzip -d [File]
  - 引数で指定されたファイルを解凍する。
  - decompress

定石

- 一般的には、配布するファイルは大量かつ大きなサイズとなる。
- なので、アーカイブで一つのファイルにまとめた後、圧縮する。
- そのため、拡張子は「.tar.gz」となる。これを「tarball」と呼ぶこともある。

圧縮解凍一度に

- 「.tar.gz」形式のファイルは、まず解凍してから、アーカイブを展開するが、面倒である。

- tarコマンドには、それらを一発で実行してくれるオプションがあるため、それを利用する。

- $ tar xfz tarball.tar.gz

  - -z　GNU Zip形式のファイルを解凍する。

ディレクトリ構造

ディレクトリ構造

- WindowsマシンでCドライブを開くと、様々なフォルダがある。
- それらフォルダには役割があり、データの種類によって格納場所が決まっている。
- 例として、Word、Excel等の応用プログラムは「Program Files」に格納されている。

ディレクトリ構造

- Linuxでもディレクトリの役割が決められており、それに準じた使い方が推奨される。
- ディレクトリ構造は、FHS（File Hierarchy Standard）という規格によって決められている。

よく使うディレクトリ

- /etc　　設定ファイル
- /boot　　起動に関するファイル
- /bin　　基本的なコマンド
- /home　ユーザのホームディレクトリ
- /sbin　　システム管理コマンド

よく使うディレクトリ

- /lib　　ライブラリ群
- /root　　rootのホームディレクトリ
- /usr　　起動時には不要な重要なファイル
- /usr/local　　ソースコンパイルでのインストール先
- /var　　ログなど、動的なファイル

ソースコンパイルによるパッケージの導入

概要

- パッケージ管理システムは便利だが、利用したいパッケージが提供されていないことがある。
- また、パッケージのバージョンが古い場合がある。
- そこで、パッケージのソースコードを自前でコンパイル、インストールすれば、上記の問題を解決できる。

ソースインストールの欠点

- 設定が面倒（逆に言えば、柔軟な設定が可能）。
- コンパイルに時間がかかる。
- 基本的に、アンインストーラはない。各所にコピーされたファイルを手作業で削除する必要がある。

GNU Hello

- 手始めに「GNU Hello」というソフトウェアをソースコンパイルでインストールしてみる。

- GNU Helloは「Hello, World!」と表示するだけのソフトウェアである。

- ソースコンパイルによるインストールを学ぶには手軽でちょうどよい。

ソースコードのダウンロード

- $ wget [URL]
  - URLで示すファイルをダウンロードする。目を通すべきドキュメント

configure

- コンパイルを実行する前に、そもそもコンパイルを行える環境が整っているかチェックする必要がある。

- また、ソフトウェアのインストール先や不要な機能を切り落とすなど、インストールに関する設定も行わなければならない。

- それらを担当するのが「configure」というスクリプトである。

configure

- $ ./configure --help
  - まずは、helpオプションを付けて実行し、オプションの一覧を見てみる。
- $ ./configure --prefix=[DIR]
  - インストール先の指定。

make

- コンパイルには「make」コマンドを使う。

- # make
  - 管理者権限が必要。
  - かなり時間がかかるので、気長に待とう。

make install

- コンパイルが完了したら、インストール（ファイルのコピー）を行う。

- # make install

  - これも管理者権限が必要。

実行

- インストール先ディレクトリの「bin」以下に「hello」というバイナリがある。早速実行してみよう。

- $ ./hello