$ man setup-dev-env

Include: CUI basic, Git, Vagrant, Node.js, Ruby, Package Management, and more...

Web サイト制作の時流に乗り遅れないために、覚えておきたい開発環境の作り方

Development Environments for Web Designers

こもりまさあき

Development Environments for Web Designers

Web サイト制作の時流に乗り遅れないために、覚えておきたい開発環境の作り方

MASAAKI KOMORI

This book is for sale at http://leanpub.com/defwd

This version was published on 2015-02-01

Contents

はじめに

この数年のデバイスの多様化にともなって Web 制作の手法も大きな転換期を迎えようとしています。Web サイトの配信対象が増えると言うことは多くの環境をサポートしていく必要があり、その答えのひとつとして Responsive Web Design のような手法も登場しました。Web 制作の主にフロント側を担当する人たちは、これまで以上に頭で考えることも手を動かすことも増えているのが現状ではないでしょうか。閲覧対象が増えるだけであればまだしも、環境変化の速さに足並みを揃えるかのようにビジネススピードも増すばかりです。

コンピュータが得意な仕事はコンピュータに任せて、人間でしかできないことに注力しなければ終わる仕事も終わりません。数年前から CSS プリプロセッサなどのツールの利用者も増えているようです。しかし、GUI のツールのバージョンアップは既にその流れについていけていないのが現状で、便利な機能を享受したければコマンドラインでの操作は必須です。その一方ではその波に乗り切れない方も多く見受けられ、それらを使える人と使えない人の差が出てきていると感じます。

「自分に合わない、もしくは必要ないから使わない」というのと、「わからない、使えない」というのでは大きな違いがあります。いまどきの Web 制作はフロントエンドとバックエンドの境界線も曖昧になりつつあります。ひとつの仕事は個人だけで完結するものの方が少なく、多くの場合はプログラマやエンジニアの方との協業になるでしょう。そういうところで共通言語ですら話せないのでは仕事はどんどんできる人の方に流れていくかもしれません。

本書はそんなこれからの新しい時代に乗り遅れないようにするため、コマンドライン操作の基本から制作環境の作り方、フロントエンド系のツールのインストールや管理方法などを紹介する書籍です。なにもバリバリとコマンドラインだけを使って制作を進めるわけではありません。作業をスムーズに進めるため、また仕事を早く終わらせるためにも、道具は適材適所で使えるようにした方が良いでしょう。技術やツールの進化によって Web サイト制作はこれまで以上に簡単にもなっているのです。

主に Web のフロントエンドを担当する皆さんは、これまでコマンドラインの操作が難しく感じていた方が多いかもしれません。コマンドライン操作は基本さえ覚えて慣れればさほど難しいものではなく、いろいろな場面で応用が効くということに気付いていただけたら。本書がそんな皆さんの一助になれば幸いです。

こもりまさあき

本書籍は「ePub/mobi/PDF」の 3 種類のファイル形式で提供していますが、LeanPub の自動生成システムを使用しているため、その体裁は基本リフロー型のレイアウトになっています。17.8cm x 23.1cm の PDF 換算で現在 300P 程度ありますが、若干レイアウト面において空白などが目に付く部分もあるかもしれません。その点あらかじめご了承いただきたく思います。

変更履歴

- 2015.02.01: 誤字脱字の修正。Chapter 7 の内容を追加（Ver. 0.4.0）
- 2015.01.15: 誤字脱字の修正。Chapter 5、6 の内容を追加（Ver. 0.3.0）
- 2015.01.09: Homebrew のアップデート方法、Chapter 4 の内容を追加（Ver. 0.2.0）
- 2015.01.05: 「Early Release」として初版発行（Chapter 1、2、3 収録）（Ver. 0.1.0）

謝辞

- コマンド入力に関する「Undo/Redo の方法」をひらいさだあき氏にいただきました。ありがとうございました。
- コマンド入力に関する「&& と; 」の違いを詳しく追記しました。@ryumu 氏、ありがとうございました。

いまどきのサイト制作とは

Web サイト制作、それもフロントエンド側にあたる作業はデバイスの多様化とともに複雑化しています。この数年で制作時に使うツールは、人によって組織によって、また担当する案件やプロジェクトの内容によっても変わってきていることでしょう。これまでのようにDreamweaver やテキストエディタだけがあればどうにかなる、という時代ではなくなってきたことを痛感している人も多いのではないでしょうか。これからの Web 制作が少し楽になるよう、そして次の時流に乗り遅れないためにも比較的新しめなサイト制作環境を紹介していきましょう。

変わり続ける Web サイト制作

Web サイト制作を取り巻く環境がこの数年で大きく変わり始めています。「世界と日本では Web サイトが全然違う」とよく言われますが、それは単純な見た目だけの話ではありません。コンテンツ配信に対する考え方や作り方も次の時代を見据えてるように思えます。デバイスの多様化にあわせて Web サイト制作を取り巻くいろいろな環境が変わりつつあることを認識しておく方が良いでしょう。

閲覧環境が変わるということ

iPhone や Android デバイスの登場以来、これまで主にデスクトップ PC だけを対象に考えていればよかった時代は終わりました。ご存知のようにこの数年のスマートデバイスの普及は目を見張るものがあります。これまでデスクトップ PC だけを対象としてきた制作会社や個人であっても、その普及率にもう目を背けることはできません。これまでデスクトップ PC だけのことを考えていればよかったのに、一気にコンテンツの配信対象が増えたも同然です。

これまでは Dreamweaver やお気に入りのテキストエディタ、それにグラフィックソフトがあればどうにかなりました。しかし、さまざまな閲覧デバイスでコンテンツを使う人たちが増えています。回線環境はバラバラになり、画面サイズもバラバラ、そんな対象に向けていかにストレスなくコンテンツを配信するか。ビジネススピードも早くなっていて、これまでのようなやり方を続けていては到底太刀打ちできないばかりか、時間ばかりが無駄に過ぎていくことにもなりかねません。

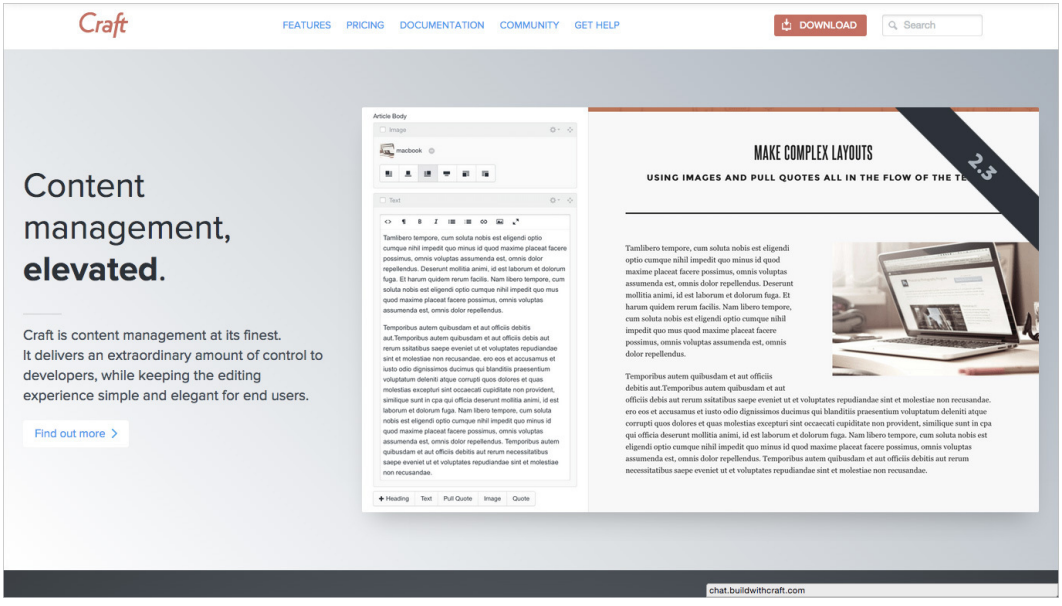
コンテンツの作り方ですら多様化

日本においては、MovableType や WordPress などの CMS を使ったサイトの作り方、CSS による見た目の実装をどうすればいいか、jQuery を使った視覚的な表現としてのインタラクションの付け方など、主にフロント側で Web 制作に関わる人たちの興味の対象はまだまだそういったところに重きがあるようにも見えます（ニーズの問題でそれが悪いわけではありません）。

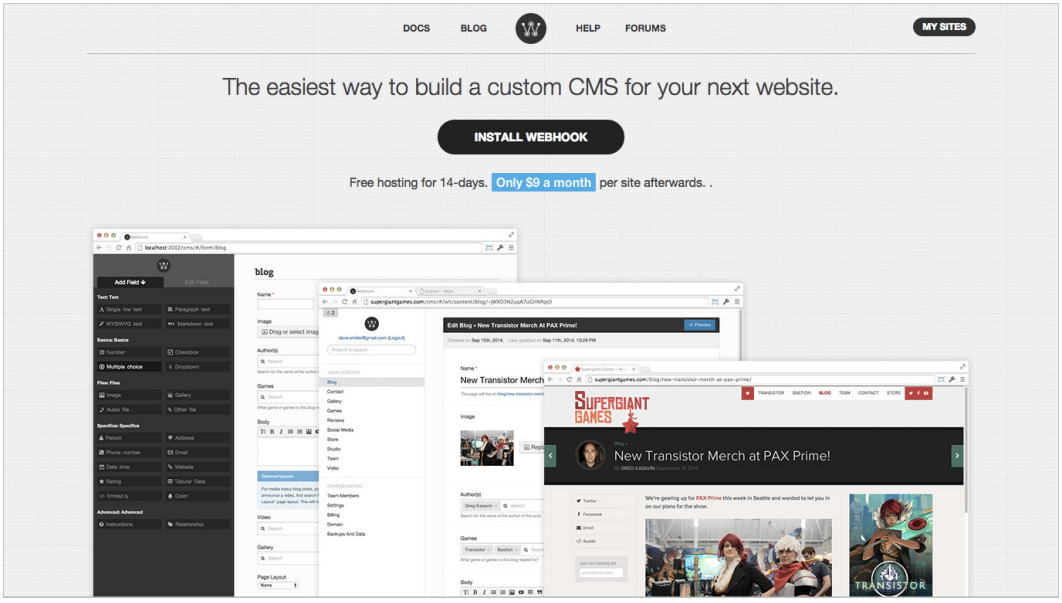
Web サイトのコンテンツが静的なものから動的な要素を含み、デバイスの多様化にあわせるかのように人々の行動までもが変化してくるようになってくると、閲覧する人のコンテキストに合わせた情報配信やリアルタイムコンテンツのニーズも出てくるでしょう。もう jQuery とそのプラグインを使って動きをつける程度の知識では無理があり、より深い JavaScript の知識が必要になっています。コンテンツを構成するのに必要な HTML や CSS ですら素の状態では書かなくなりつつあるのです。

CMS にしても「Craft」や「Webhook」の方がカスタムフィールドベースでサイト制作ができるだけでなく、「Twig」や「Swig」のテンプレート言語と HTML を使って見た目をコントロールがしやすいかもしれません。また場合によっては、CMS をがんばって使うよりは Static Site Generator の方が都合が良いこともあるでしょう。

バックエンドを含んだ Web サイト制作においては、長い間「LAMP（Linux / Apache / MySQL / PHP）」の構成がもてはやされているようにも見えますが、Web サイトや Web アプリケーションを作る環境はそれだけではありません。Ruby を使ったフレームワークの「Ruby On Rails」など有名ですが、近頃では NodeJS をバックエンドとして動かす「MEAN（MongoDB / Express / Angular / Node）」のような構成での運用事例も出てきています。



PHP で動作する CraftCMS



node.js で動作する Webhook

mean.io は、MEAN で作り始めるためのフレームワーク

Web が Web サイトでなくなる？

さらにこの数年で REST (Representational State Transfer) インターフェイスを介して、HTTP の URI ベースでデータをやりとりすることも増えているようです。従来の CMS であってもこの REST を介してデータをやりとりできるようになりつつあります。極端な話をすれば、CMS は完全にコンテンツ管理だけをおこなうだけでよく、URL の書き換えを含めたフロント側は JavaScript のフレームワークでコントロールすることもできます。

REST のような API（Application Programing Interface）によるデータのやりとりが理解できると、いざ自分が何か作る際もバックエンドのシステムを自分で用意する必要もありません。「BaaS（Backend as a Servive）」を使えば、データベースや認証の仕組みなどもそれを利用すれば終わりです。ホスティングも含めていろいろな取り巻く環境が大きく変化していることを認識しておいた方が良いですね。

Firebase や Parse のような BaaS を使えばバックエンドはおまかせ

いまはまだこれまで通りのやり方で良いでしょうが、今後は主に Web サイトのフロント側を担当すると言ってもさまざまな環境下でのサイト制作をおこなう機会が増えるかもしれません。これまでのやり方を続けていくか、少し先を見ながらのんびりでも対応できる術を身につけるか、それは皆さん次第です。

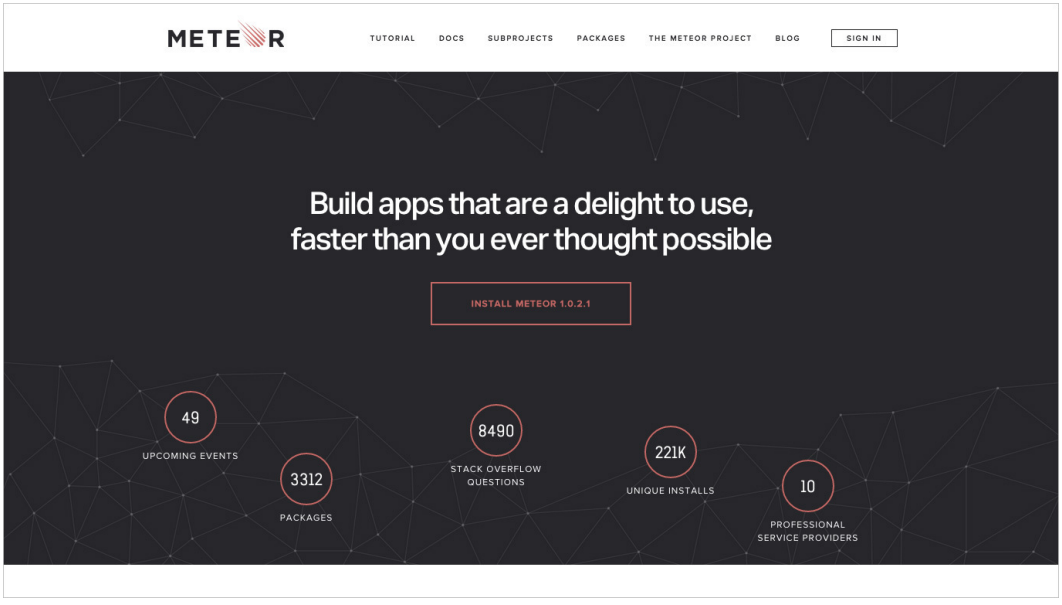
最新の制作ツールはコマンドラインから

いまどきの Web 制作ではフロント側の実装に関係することだけに絞っても、規模の大小にかかわらず実装に関わる負荷が増えています。やることが増えた分、コンピュータが得意なことは任せた方が楽になります。しかし、そういったツールはコマンドラインでしか使えないことが多いのです。

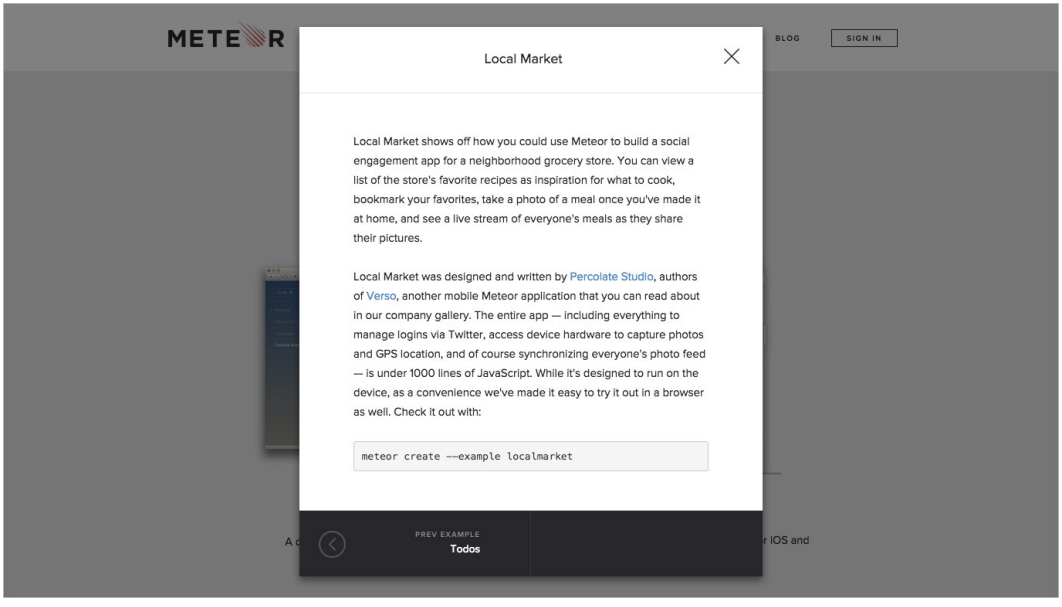
GUI だけではどうにもならない時代に

より直感的に操作できるようにと登場したはずの GUI ですが、今の時代それに逆行するかのように制作ツールに関しては CUI での動作が中心になっています。これまでは GUI の制作ツールの年に一回のアップデートでどうにかなったのかもしれません。新しい技術への対応などもそうですが、もはや日進月歩どころの進化ではなく常に新しいバージョンが提供されるような時間軸で動いてます。知らなければいつまでもその便利さを享受できません。

まだまだ日本での注目度は低いようですが、「Meteor」はリアルタイムで動作する Web アプリケーションを簡単に作るためのフレームワークです。コマンド操作で必要な機能やパーツを追加して HTML や CSS、JavaScript を書いていけば、Web とネイティブで動くハイブリッドなアプリケーションをも作ることができます。「そんな便利なものがあるのならこれを使おう」と考えても、これを動作させるにはやはり node.js の環境が必要となり、デモアプリはおろかサンプルのチュートリアルですら動かすことはできないのです。



リアルタイム Web のためのフレームワーク、Meteor



コマンド操作ができなければデモすら動かせない

ライブラリのダウンロードもコマンドライン

JavaScript のライブラリなどはこれまで.zip などでファイル一式が提供されていることが多かったので、それをダウンロードしてきて任意の場所に保存すれば使うことができました。しかし、ここ最近ではコマンドラインでのインストール方法や GitHub のリポジトリへのリンクだけしか書いてないこともあります。

いざサイト制作に必要な道具一式を揃えて作り始めようにも、サイトを駆けずり回ってかき集める人とコマンドラインで一瞬で手元に用意できる人ではそのスタート地点からして違ってきています。たとえば、以下はフレームワークの Bootstrap を手元に用意するコマンドです。これを実行すればほんの数秒で Bootstrap だけでなく、その動作に必要な jQuery までも含めてダウンロードが終わります。

```
$ bower install bootstrap
```

次のコマンドは、WordPress の最新版のファイル一式をダウンロードできます。わざわざサイトに行く必要もないだけではなく、設定からデータベースの作成まであとひとつふたつのコマンドを打つだけで WordPress のサイトを立ち上げることができるのです。

```
$ wp core download --locale=ja
```

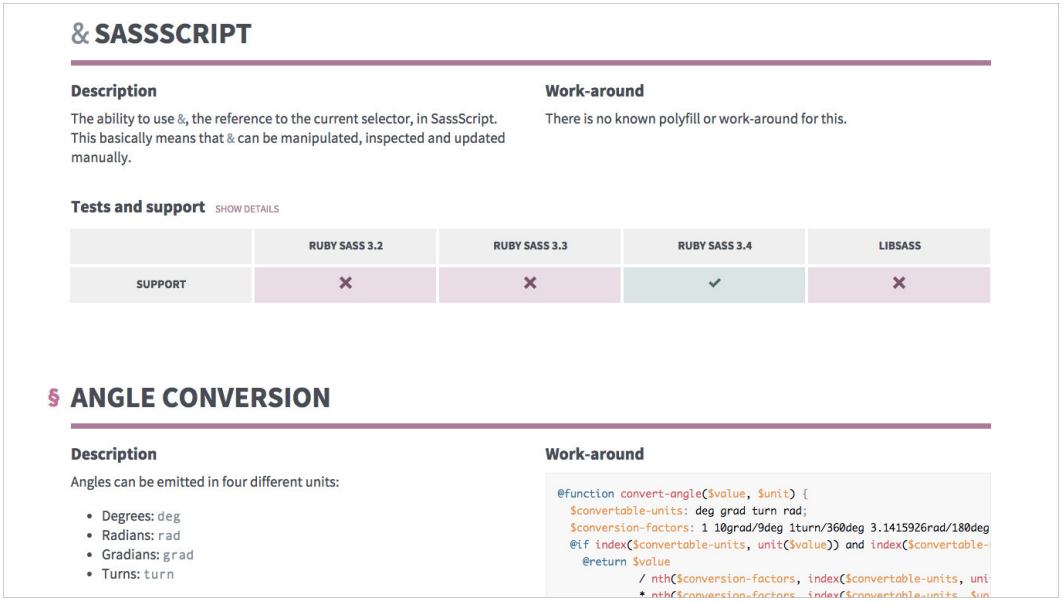
これだけでは一見ほんの些細なことのようにも思えます。しかし、コマンドで操作できるものは自動化の対象になります。使えるかどうかは仕事全体の生産性にも影響を与えるでしょう。ある人は 3 日かかってサイトを作る、しかしある人は 1 日もあればできるとなれば雲泥の差です。仮に個人事業主のような立場であれば時間単価が大きく変わってきます。

コマンド操作だけができれば解決

数年前から「深く使うことはなくても、せめて node.js と Ruby ぐらいは入れておきましょう」と言い続けてきましたが、いまどきの Web 制作をよりスムーズにおこなうためにはもうコマンドラインの操作は必須です。今さら時代に逆行していると言われればそうですが、便利なツールが GUI では提供されることの方が少ないのですから仕方ありません。

CSS プリプロセッサやタスクランナーのようなものは GUI のソフトウェアから実行することができます。しかし、これらのツールのバージョンアップもまた異様なほどの速さです。CSS プリプロセッサの「Sass」の利用者が多いようですが、Sass はバージョンや変換エンジンに

よって使える機能が異なります。GUI のソフトウェアのバージョンに左右されてしまうと、いつまでも便利な機能が使えないかもしれません。



Sass はバージョンやエンジンで使える機能が違う

今後何かを始めるにあたっては、コマンドラインからしかインストールできないソフトウェアを使わなければならないこともあるでしょう。フロントエンドとバックエンドの境界線があいまいになりつつある中、その基本操作と簡単な仕組みさえ覚えておけばいろいろなシーンで応用が効きます。「それ知らないです」「使っていません」では、仕事そのものがなくなるかもしれません。仕事はできる人の方に流れるものです。

ローカルで Web サイトを動かすには？

HTML と CSS、jQuery でのインタラクションが付与された静的なページであれば、ローカルで制作中の「～.html」のファイルを Web ブラウザで開けば動作しているかどうかを確認できます。しかし、動的に外部のリソースを取得するタイプのサイトを作るとなるとそうはいきません。時には「file://～」ではじまるアドレスでは表示確認ができないこともあるでしょう。当然、Ruby や PHP のようなプログラムが介在する Web サイトでも同様です。

OS X なら実は簡単!?

たとえば、CMS を使おうとすればそれを動作させるためのサーバ環境が必要です。OS X にはあらかじめ Web（HTTPD）サーバとして有名な「Apache」、プログラムの実行に必要な「PHP」「Ruby」などが含まれており、それらを有効化するだけで新たにテスト環境を用意せずに利用できます。しかし、それらを動かしたくても仕組みがわからないことにはできません。わからないからといって、簡単にローカルのテスト環境が用意できるオールインワンパッケージになったソフトウェアが良いかというとそうでもないのです。

MAMP や XAMMP の問題

バックエンドの仕組みに詳しくない方は「MAMP」や「XAMPP」のように、あらかじめ Web サーバと Perl、PHP、MySQL などがひとつのパッケージとなったソフトウェアを使っているでしょう。必要なものがパッケージされたこれらは簡単に導入できて大変便利な反面、実際に動作するサーバの構成とは異なるがゆえの問題も出てきます。

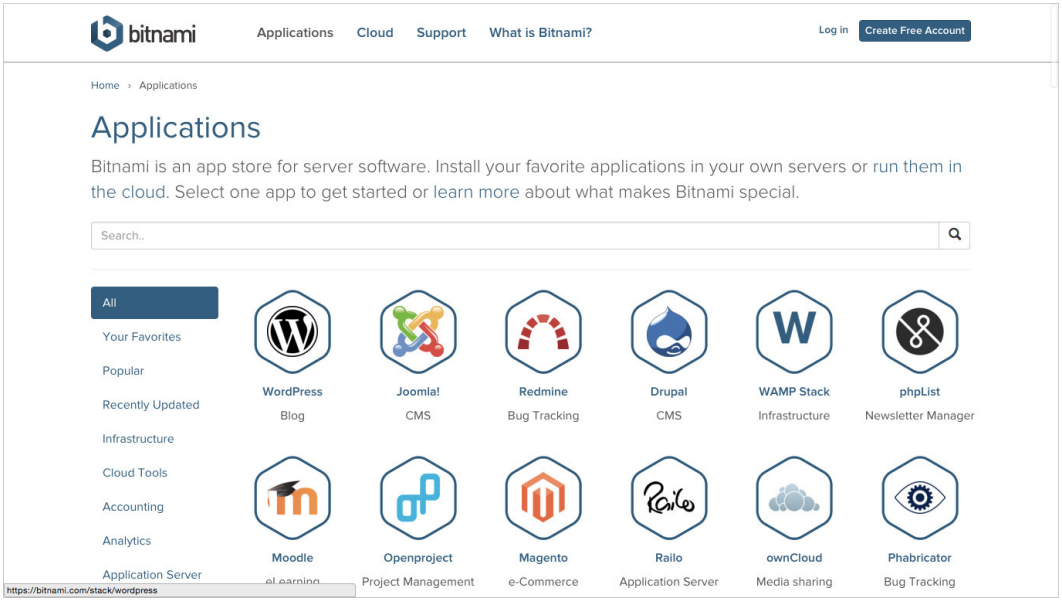
Web サーバにも PHP にも MySQL にもそれぞれ設定があります。エラーが出た場合、OS 側の問題なのかアプリケーションの問題なのか切り分けができなくなり問題の解決に時間がかかってしまう事にもなりかねません。MAMP は OS X Yosemite へのアップグレードによって正常に起動できないという問題が起きた方もいらっしゃるようですが、特殊な環境であることがむしろ困る場合もあるのです。

最近では Web サーバも「nginx（エンジンエックス）」の人気が高まっています。こういった Web サーバの特徴や仕様・設定方法を知らないままでいては、ホスティングされたサイトの動作設定の変更はもちろんのこと、テスト環境をあわせて作ることもできません。組織の一員であれば誰かがやってくれるでしょうが、個人事業主などで受託をしている場合では困ります。HTML5 のアプリケーションなどバックエンドのシステムがローカルに不要なサイトは、JavaScript のツールを使ってローカルサーバを立ち上げる方が簡単です（いざとなったらそのまま外部公開もできます）。

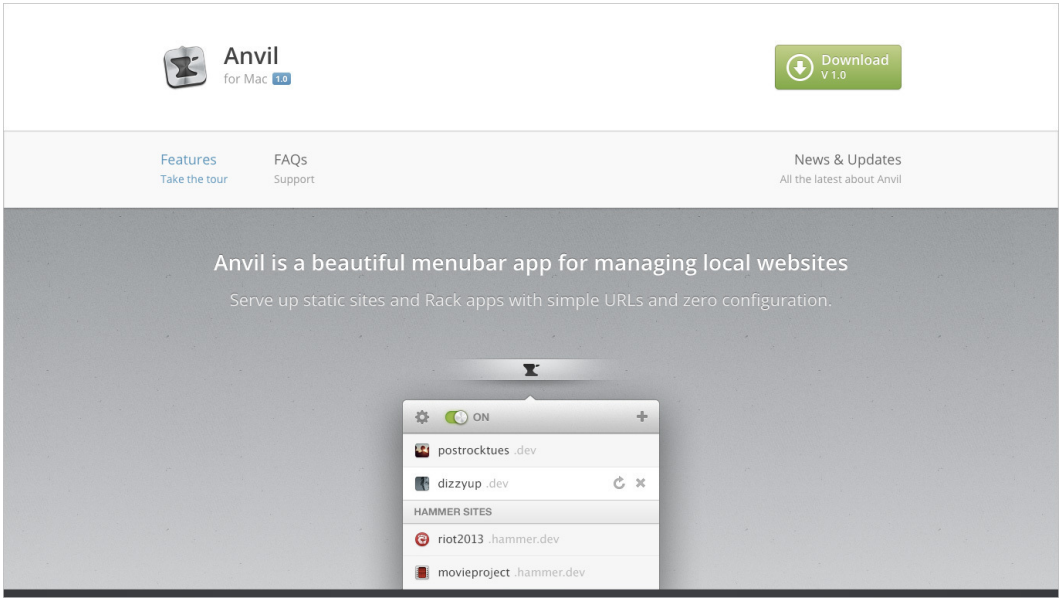
```
$ ss
```

ツールのインストールは必要ですが、このたった 2 文字のコマンドを入力するだけで SPA（シングル・ページ・アプリケーション）にも対応可能なサーバを起動できます。

新しい技術に対応した環境を GUI で用意するなら「Bitnami Stacks」を使えば簡単です。静的なサイトの確認程度なら「Anvil」を使えば、ディレクトリがどこだろうがそこをルートディレクトリとしてローカルサーバを起動できます。



Bitnami であれば最新の環境も GUI で



任意のディレクトリをルートにするなら Anvil

新しい時代に対応するには

このように本書の Chapter 07 では、フロントエンド側の制作に役立つツールだけでなく、JavaScript を使ったローカルサーバの起動方法、ローカルマシンの制作サイトの状態を一時的に外部に公開する方法や PaaS（Platform as a Service）を使ったテスト環境の公開も紹介します。本書の後半では仮想環境をサーバの OS から選択して構築できる「Vagrant」というソフトウェアを使い、OS の中を極力汚さないで済むような開発・テスト環境を動かしてみましょう。

仮にきっちりと分業体制が敷かれていたとしても、制作するコンテンツによってはフロントエンドとバックエンドの境界線は曖昧になっています。そういった時流に逆らうことなく新たな潮流にも少しずつでも対応できるよう、いまどきのサイト制作をおこなうための環境を少しは理解しておいた方が良いでしょう。本書は、ターミナルを使ってコマンド操作を極めるのではなく、コマンド操作の基本を覚えながらさまざまなツールをすんなり導入し使えるようになることを目的としています。

ターミナルの操作に慣れよう

CUI（Character User Interface）/ CLI（Command Line Interface）は、テキストでコマンドの文字列を入力して実行します。一見難しそうに感じますが、日常的に使うコマンドは限られています。GUI の画面でボタンを押す代わりに、テキストで命令を与えてるだけだと考えれば決して怖いものではありません。いまどきの Web 制作のワークフローを効率化するには、この CUI でしか使えないソフトウェアが必要なのです。いまのうちからターミナル（Terminal.app）の操作に慣れておきましょう。

シェル？

ターミナルを起動すると OS X では「bash（バッシュ）」と呼ばれるシェルプログラムが起動します。このシェル（Shell）の基本的な仕事は、他のプログラムを起動することです。

主なシェル

UNIX 系の OS の標準のシェル（ログインしたときに起動されるシェル）は bash であることがほとんどのようです。実はシェルにはいくつかの種類があり、自分の好きなシェルを指定して使うことができます。それぞれのシェルは独自に備えた機能であったりコマンドの補完のような使い勝手が異なるため、エディタと同じように人によって使うシェルの好みがあります。

- sh（Bourne shell）
- bash（Bourne-Again shell）
- csh（C Shell）
- tcsh（TENEX C shell）
- zsh（Z Shell）

コマンドの説明は後ほどおこないますので、まずは Finder からターミナル（Terminal.app）を起動し、表示されてる「$」のあとに下記のコマンドを入力してリターンキーを入力してみましょう。「cat」と「/etc/shells」の間には「（半角スペース）」を入れてください。

> cat コマンドを実行

```
$ cat /etc/shells（リターンキー）
```

ターミナルを起動してコマンドを入力

コマンドを実行すると以下の内容が表示されるでしょう。

> cat コマンドの実行結果

```
/bin/bash
/bin/csh
/bin/ksh
/bin/sh
/bin/tcsh
/bin/zsh
```

コマンドを実行した結果として、ディスクの「/etc」の中にある「shells」ファイルの内容が画面上に出力されたのです。ここにリストされたシェルは、あらかじめ OS X にインストール済みのものです。シェルは「他のプログラムを起動すること」が仕事ですから、標準のシェルである bash からこれらを起動することもできます。bash から「zsh」と打ってリターンキーを押すと zsh が起動します。

> zsh を起動

```
$ zsh（リターンキー）
```

コマンドを実行すると、zsh が起動して以下のようにターミナルの表示が変わります。

> zsh のプロンプトの表示

```
マシン名%
```

これは「bash が起動しているうえにさらに zsh が起動している」状態です。

【zsh】
【bash】

続けて「ps」コマンドを実行してみましょう。自分自身が起動しているプロセスが表示されて bash と zsh が同時に起動していることがわかります。

> ps コマンドの実行

```
% ps
```

> 起動中のプロセス

```
 PID TTY           TIME CMD
9782 ttys000    0:00.01 -bash
9867 ttys000    0:00.02 zsh
```

起動した zsh を終了するには「control＋d」のショートカットを入力します。ターミナルでプログラムを実行したり、リモートのサーバに SSH（Secure SHell）でログインしたりするようになると、いたるところで以下の 2 つのショートカットを使います。いまのうちに覚えておきましょう。

- **control＋D**: ログアウト（シェルの終了、リモートサーバからログアウト）
- **control＋C**: プログラムを終了する

本書は OS X の標準シェルである「bash」を使って解説を進めます。

シェルの概念や機能について深く知りたい場合は、以下の記事をご覧ください。
（参考資料）シェルの概念と機能

覚えておきたいシェルのコマンド

ここからは実際にコマンドを入力・実行して、ターミナルの操作に慣れていきましょう。各コマンドを入力した後は「リターンキー」で実行します。

> **注意** 本ページ以降コマンドの実行文が多く含まれますが、ところどころ現れる「\（バックスラッシュ）」は通常記述できない文字を表記する際に用いられます（参考: バックスラッシュ）。本文のコードやコマンド中でバックスラッシュが含まれている場合は、その直後の半角スペースなどをエスケープしているか、折り返しの含まれる長い行のコマンドをその部分で改行していることを表します。本書は Leanpub による電子書籍を自動生成しているためフォーマットによって任意の場所でエスケープされています。

自分の居場所を表示する

まずは基本中の基本のコマンドを使って、自分の現在の居場所（現在の作業ディレクトリ）を表示してみましょう。

> 現在のディレクトリを表示

```
$ pwd
```

ターミナルの起動直後は自分のホームディレクトリにいますので、コマンドを実行するとターミナルの画面には自分のホームディレクトリのパス（ローカルマシン内での位置）が表示されます。

> pwd の実行結果

```
/Users/自分のアカウント名
```

「pwd」は「Print Working Directory」の略です。ターミナルで実行するコマンドの名前は、これからおこないたい処理内容の頭文字を取った略語がほとんどです。グラフィカル・ユーザー・インターフェイスの画面でいえば「pwd」と書かれたボタンを押してると考えてください。

ディレクトリの内容をリストする

自分がいるディレクトリにあるディレクトリやファイルをリストしてみましょう。現在の作業ディレクトリは、さきほど「pwd」で表示された自分のアカウントのホームディレクトリです。「ls（小文字のLとs）」と入力してください。

> ls コマンドの実行

```
$ ls
```

ルートディレクトリにあるディレクトリやファイルがリストされましたか？

> ls の実行結果

```
Applications  Downloads  Music
Dropbox  Pictures  Desktop
Public  Library  Documents  Movies
```

任意の場所をリストするには、以下のように ls の後に半角スペースをいれて場所（パス）を入力します。

> 場所を指定してリスト

```
$ ls 表示したい場所
```

たとえば、システムのルートディレクトリ「/」にあるものを見る場合はこうなります。

> '/' の内容をリスト

```
$ ls /
```

OS X でルートディレクトリをリスト表示するとこのような感じになっているのではないでしょうか。

> 'ls /' の実行結果

Applications	Users	dev	private
Library	Volumes	etc	sbin
Network	bin	home	tmp
System	cores	net	usr
TSSleepHandlerHelp	opt	var	

日頃使っている GUI の Finder からは見えてない領域が多いことがわかります。このディレクトリ構成は、UNIX 系のマシンでは大体似たり寄ったりでおなじみのものです。この先リモートにある Linux サーバなどにログインする機会もあるでしょうから、これらのディレクトリがどのような役割になっているか簡単ですが紹介しておきます。

- /bin（基本コマンドが入っている）
- /etc（設定ファイルの多くは基本的にここに）
- /home（ユーザーのホーム。OS X では /Users なのでここは空）
- /sbin（システムの管理コマンドなど）
- /tmp（一時領域）
- /usr（各種コマンド、プログラムなどが入っている）
- /var（ログや Web サイトのデータなど、変更されるデータの格納領域）

コマンドのオプションを指定する

ターミナルから実行できる各種コマンドは、「-（ハイフン）」や「--（ハイフン ×2）」で始まるオプション指定が用意されているものがあります。以下のように半角スペースに続けて「-l（小文字の L）」を加えて「ls」コマンドを実行してみましょう。

> '-l' を付けて ls コマンドを実行

```
$ ls -l
```

先ほどのように単純に「ls」を実行しただけではディレクトリやファイル名が羅列されただけだったものが、この「ls -l」のように「-l」オプションを付けることでパーミッションや所有者とグループも一緒に画面内にリスト表示することができます。

「-a」オプションを付ければ、不可視ファイルも表示します。

> '-a' を付けて ls コマンドを実行

```
$ ls -a
```

オプションはまとめてすることもできます（`ls -l -a` でも可）。

> 複数のオプションを指定して実行

```
$ ls -la
```

冒頭で紹介した「ps」コマンドもオプションを付けて実行すれば、起動中のプロセスをさまざまな形で閲覧できます。次のコマンドを実行すると、システムが実行しているプロセスをはじめとして起動中のプロセスがすべて確認できるでしょう。

> プロセスの表示

```
$ ps -ax
```

どうでしょうか？GUI で PC を操作していると表向きには見えないだけで、実は裏側ではこのように多くのプロセスが起動して動いているのです。本書を読み進めていく過程では、至る所でコマンドラインを使ってツールを起動します。動いてるのかどうかわからない時などは、このようにして調べることができると覚えておきましょう。

コマンドの使い方やオプションが知りたいときは「man」コマンドを実行するか、コマンドによっては「-h」や「--help」「help」のオプションと一緒にコマンド実行すればヘルプ画面を表示することができます。

> ls コマンドのマニュアルを表示

```
$ man ls
```

マニュアルが表示されたら、「リターンキー」「矢印キーの上下」「F キー（forward）」「B キー（backward）」を使って表示位置を進んだり戻したりすることができます。変なキーを押してしまったら「esc キー」を。マニュアル画面を終了するには「Q キー」を入力します。

作業ディレクトリを移動する

ターミナルでは、現在自分がいるディレクトリを基準にして各種コマンドを実行します。現在の作業ディレクトリを変えてどこか別のディレクトリに移動するには、「cd」コマンドを実行します。言うまでもなく「Change Directory」の略です。

> デスクトップに移動する

```
$ cd ~/Desktop
```

上記コマンドでデスクトップに移動します。「～（チルダ）」は、自分のホームディレクトリ（$HOME）を表しますので覚えておきましょう。「$HOME」のように $ ではじまる文字列は、環境変数として登録されています。次のようにコマンドに組み合わせて使うことも可能です。

> 環境変数を使った移動

```
$ cd $HOME/Desktop
```

「cd」だけを実行する、または「cd ～」を実行すれば自分のホームディレクトリに戻ります。

> '～' を付けてホームに移動

```
$ cd ~
```

一つ上の階層に移動したい場合は、「cd ..（ピリオド ×2）」を入力しましょう。

> ひとつ上の階層に移動

```
$ cd ..
```

Web サイト制作で相対パスを指定するのと同じですね。このようにコマンド操作だけで場所を移動したり、ディレクトリの中身を確認したりするのは簡単なことです。シェルの扱いに慣れてくると、Finder を使って場所を移動しディレクトリを開くといった操作すらもどかしく感じるかもしれません。

> ターミナルでは日本語の入力もできますが、移動などをよりスムーズにおこなうためにも英数字を使ったディレクトリ名にしておく方が良いでしょう。日本語の OS X の Finder 上で「デスクトップ」「書類」と表示される OS の標準ディレクトリは、それぞれ「Desktop」と「Documents」でアクセスできます。

入力補完を利用する

「cd ～/Desktop」のように入力する文字列が長くなればなるほど自分で入力すれば記述ミスが発生しますし、ディレクトリが深くなればなるほど（ファイル名が長くなればなるほど）入力に時間がかかってしまうでしょう。そこで時間がかかってしまうのであれば、GUI で操作した方が早いということになってしまいます。

コマンドの入力途中で「tab キー」を押せば、その先の内容を補完する機能がほとんどのシェルに用意されています。

> 'cd ～/D' を入力

```
$ cd ~/D（tab キー）
```

「cd ～/D」まで入力して tab キーを 2 回押せば、bash の場合は「Desktop/　Documents/　Downloads/」のようにルートディレクトリ以下にある「D」ではじまるディレクトリが表示されます。

> 入力内容を追加

```
$ cd ~/Des（tab キー）
```

デスクトップに移動したい場合は、その後に続く「es」ぐらいまで入力して再度「tab キー」を入力するとその先の入力が補完されて自分で入力する必要はありません。デスクトップにあるディレクトリ名やファイル名がわからなくても、補完された後に再度 tab キーを 2 回押せばその次の候補が表示されます。

> ディレクトリ内をさらに調べる

```
$ cd ~/Desktop/（tab キー）
```

大量にリストされた場合は「:more」が画面内に表示されます。この表示を終了するには「Q キー」を 1 度入力しましょう。ファイル名の冒頭の文字を数文字入れれば、それが含まれるものに絞り込まれるので、また tab キーを使って補完していけば入力は最低限で終わります。

> Finder の「ディレクトリへ移動」メニューでも「tab キー」による補完が可能です。

ヒストリー（入力履歴）を利用する

ターミナルでの操作は、何度も何度も同じコマンドを入力することもあるでしょう。入力補完があるとはいえ、同じ内容を何度も何度も入力するほど面倒なことはありません。bash（やそれ以外のシェルも含む）は、一度実行した結果をヒストリーとして保存しています。

> ヒストリーの表示

```
$ （矢印キーの上下）
```

上矢印のキーを押していけば、過去にそのシェルで入力したコマンドを遡ることができます。行き過ぎたら、下矢印キーを入力しましょう。このコマンドの実行履歴は、自分のホームディレクトリの直下に「`.bash_history`」という不可視ファイルで保存されます。

> bash では矢印キーの上下以外での選択以外に、「control＋R」キーを押してから任意の文字列を入力し、ヒストリーをインクリメンタルサーチすることもできます。

コマンド行のキャレットの移動

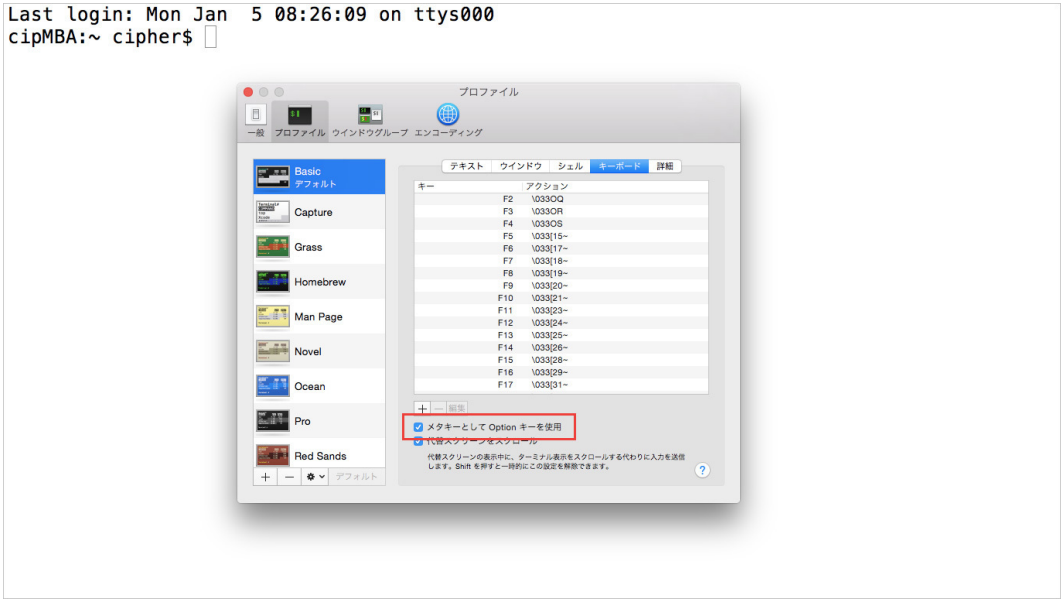
ターミナルでは、ここまで紹介したようにして任意のコマンドをテキストで入力します。実行するコマンドにはオプションだけではなく、ディレクトリ名、ファイル名などを付け加えるので、長文のコマンドで打ち間違えなどがあると修正箇所まで戻るのが大変です。たとえば、自分のホームディレクトリ以外で下記のコマンドを行末まで入力してから間違いに気付くことがあります。

```
$ cd Desktop/folder-name/filename
```

正確には「`~/Desktop/folder-name/filename`」のようにチルダが必要です。ここで一文字ずつ入力内容を消去したり、「矢印キー（←→）」や「control＋F」「control＋B」で一文

字ずつ移動するのは面倒です。こういう時に備えてコマンド行の中のキャレットの位置を移動するショートカットを覚えておくと良いでしょう。bash では、「control＋A」でコマンド行の先頭、「control＋E」でコマンド行の最後に移動できます。

ターミナルの環境設定で「メタキーとして Option キーを使用」のチェックボックスを入れておくと、「option＋左矢印（または option＋F）」で単語をひとつずつ飛ばしてキャレットを後ろに移動、「option＋右矢印（option＋B）」で逆に前方に単語をひとつずつ飛ばしてキャレットを移動できます。



ターミナルの環境設定でメタキーを有効に

入力しかけたコマンドを一旦キャンセルするには「control＋C」を入力します。これで入力した内容が消去されます。「delete キー」で消すより簡単です。

> 入力をキャンセルする

```
$ cd Desktop/folder-nam（control ＋ C キーを実行）
```

入力途中のコマンドを一旦取り消してまた再び入力するのであれば、「control＋U」と「control＋Y」の組み合わせを覚えておくとよいでしょう。「control＋U」で一度取り消した内容は、「control＋Y」で復活させることが可能です。

> 入力を取り消す

```
$ cd Desktop/folder-nam（control ＋ U キーを実行）
```

> 取り消した内容を戻す

```
$ （control ＋ Y キーを実行すると内容が戻る）
```

Finder でディレクトリを開く

任意のディレクトリを OS X の Finder 上に開くには「open」コマンドが使えます。「.（ピリオド）」は現在のディレクトリを表しますので、下記のコマンドを実行すると Finder で現在の作業ディレクトリが開きます。

> 現在のディレクトリを Finder で表示

```
$ open .
```

OS X には Finder からは直接開けないディレクトリもあります。開きたい場合は、Finder の「ディレクトリへ移動」メニューを使わなくてもターミナルから直接開くことができます。

> 不可視のディレクトリを表示する

```
$ open ~/Library/Application\ Support/
```

ディレクトリ名やファイル名に半角スペースが含まれるものは「\ 」という風に「バックスラッシュ+半角スペース」の入力が必要です。これもタブキーによる入力補完を利用すれば簡単です。

ファイルの内容を表示する（テキストファイル）

テキストファイルの中身などを確認したい場合は、「cat」や「more」「less」といったコマンドが使えます。

「cat」コマンドでファイル名を指定すれば、そのファイルの内容がすべて画面内に出力されます。「cat」コマンドの本来の用途は複数ファイルを繋げる（concatenate）コマンドですが、単一のファイルの全体をパッと見たいといった用途にも使えます。コマンドを実行すれば、画面内にファイル内容のすべてが出力されて終了します。長いテキストファイルが出力されたとしてもターミナルの画面をマウスカーソルで遡れば内容は確認できます。

> cat コマンドでファイルを表示

```
$ cat ファイル名
```

長いテキストファイルの内容を順追って見たい時は「more」または「less」コマンドが便利です。いずれもファイルを開いて画面をスクロールさせながら内容を確認できます。「more」や「less」は、「/」や「?」を入力してから単語を入力し「return」キーを押して検索したり、ハイライトするといった使い方もできます（more と less で挙動は違います）。これらのコマンドはそのままでは起動したままになるので、終了するには「Q キー」を入力しましょう。

> more コマンドでファイルを表示

```
$ more ファイル名
```

> less コマンドでファイルを表示

```
$ less ファイル名
```

ファイルの拡張子によっては「open」コマンドを使って直接 GUI のアプリケーションで開くこともできます。

> ファイルを任意のアプリケーションで開く

```
$ open ファイル名
```

ログファイルなど長いファイルの最後の方だけ見たい場合は「tail」を使うと便利です。「tail」コマンドはファイルの最後の行から 1 画面分を画面上に出力します。

> tail コマンドでファイルを表示

```
$ tail ファイル名
```

画面の内容をクリアする

コマンド操作を続けていくと、ターミナルの画面内にはどんどん文字列が表示されていきます。現在表示されている画面を綺麗にしたい場合は「clear」コマンドを使うとスクリーンが一画面分スクロールして真っ新の画面になります。「clear」コマンドは「control＋L」のショートカットが割り当てられています。

> clear コマンドを実行

```
$ clear
```

スクロールしているだけなので、画面をスクロールすれば直前までの画面に戻れます。

新規テキストファイルを作成する

空の新規ファイルを作るには「touch」コマンドを使います。本来の用途は既存のファイルのタイムスタンプを変更するものですが、ファイルが存在しない場合は新規の空ファイルが作成されます。以下のコマンドを実行すると現在の作業ディレクトリに、真っさらの「sample.txt」というテキストファイルができるでしょう。

> 'sample.txt' ファイルを作成

```
$ touch sample.txt
```

新規ディレクトリを作成する

新規でディレクトリを作成する場合は「mkdir」コマンドを使います。言うまでもなく「MaKe DIRectories」の略です。

> 新規ディレクトリの作成

```
$ mkdir ディレクトリ名
```

任意のディレクトリ名を付けて実行すれば、現在の作業ディレクトリに新しいディレクトリが作成されます。場所を指定してディレクトリを作りたい場合は、そこまでのパスを入力してディレクトリ名を指定します。

> 場所を指定してディレクトリを作成

```
$ mkdir ~/Desktop/ディレクトリ名
```

「mkdir」コマンドでは、半角スペースを空けて作りたいディレクトリ名を列挙すれば複数のディレクトリを一度に作成可能です。

> 複数のディレクトリの作成

```
$ mkdir ディレクトリ A ディレクトリ B
```

「-p」オプションを付け加えれば、階層構造をもったディレクトリも直接作れるので覚えておきましょう。

> 階層付きでディレクトリを作成

```
$ mkdir -p 親ディレクトリ名/子ディレクトリ名
```

これらを組み合わせれば、以下のように一回のコマンドで「projects」ディレクトリ内に「images」「css」「js」のディレクトリをあらかじめ用意するといったことができますね。

> 階層付きのディレクトリを複数作成

```
$ mkdir -p projects/images projects/css projects/js
```

ファイルやディレクトリを移動する（リネームする）

ファイルやディレクトリを移動する場合は、「mv」コマンドを使います。言わずもがな「MoVe files」です。移動する対象と移動する場所を対で指定します。

> mv コマンドを実行

```
$ mv ファイル名 移動する場所
```

ファイルをデスクトップに移動する場合は以下のようになります。移動先がディレクトリであれば特にファイル名を指定する必要はありません。

> デスクトップにファイルを移動

```
$ mv ファイル名 ~/Desktop/
```

ファイル名も指定したい場合は、移動先でのファイル名を指定しましょう。

> ファイル名を指定して移動

```
$ mv ファイル名 ~/Desktop/新しいファイル名
```

「mv」コマンドは、ファイルの移動だけでなくファイル名を変更も可能です。同一階層で「mv」コマンドを実行すれば、単純にファイル名がリネームされます。以下の例は、「a.txt」が「b.txt」に変更されます。

> ファイル名の変更

```
$ mv a.txt b.txt
```

ファイル同様、ディレクトリを移動する場合も「mv」コマンドを使います。

> ディレクトリを移動

```
$ mv ディレクトリ名 移動する場所
```

簡単ですね。ディレクトリ名を変えたい場合も「mv」コマンドを使いますので覚えておきましょう。

ファイルやディレクトリをコピーする

ファイルをどこか別の場所にコピーしたり、別名で保存しておく場合は「cp」コマンドを使います。言うまでもなく…、です。

> ファイルをコピーする

```
$ cp ファイル名 コピーする場所
```

「mv」コマンド同様に移動先でのファイル名を指定することもできます。

```
$ cp ファイル名 ~/Desktop/新しいファイル名
```

同じ階層で「cp」コマンドを実行すれば、同じものが別名で複製されます。慣れないターミナルでテキストファイルを編集する際は、あらかじめファイルを別名で複製しておいて作業するのが安全です。

> ファイルの複製

```
$ cp ファイル名 新しいファイル名
```

「cp」コマンドはファイルだけでなくディレクトリのコピーも可能ですが、ディレクトリの場合は「cp」コマンドをそのまま実行してもエラーになります。ディレクトリの場合は「-R（-r）」オプションが必要です。

> ディレクトリのコピー

```
$ cp -R ディレクトリ名 移動先/ディレクトリ名
```

OS X にはファイルやディレクトリのコピーする「ditto」という別のコマンドが用意されています。'ditto' コマンドは、ディレクトリをコピーする時もオプション指定が不要ですのでこちらの方が簡単ですね。

> ditto コマンドによるコピー

```
$ ditto ディレクトリ名 移動先/ディレクトリ名
```

ファイルやディレクトリを消去する

ファイルやディレクトリを削除する場合は「rm」コマンドを使います。「ReMove」で覚えると良いでしょう。

「rm」コマンドを実行すると Finder のゴミ箱に入るわけではありません。実行すれば問答無用で、何事もなかったかのようにファイルやディレクトリは消え去りますので注意が必要です。

> rm コマンドでファイルを消去

```
$ rm ファイル名
```

中身のあるディレクトリの場合は「-R（-r）」オプションを付けて実行しましょう。

> ディレクトリを消去

```
$ rm -R ディレクトリ名
```

いきなり消されるのが怖い場合は、「-i」オプションを付けて実行することで確認のダイアログを出すことができます。

> '-i' オプション付きで rm コマンドを実行

```
$ rm -i ファイル名
```

コマンドを実行すると「Remove ファイル名？」と聞かれますので、消去する場合は「Y」キー、キャンセルする場合は「N」キーを入力します。うっかり「return」キーを押しても大丈夫です。キャンセルされます。

> 'rm -i' による確認

```
Remove ファイル名？(y か n を入力)
```

rm コマンドは慣れるまで慎重に実行しましょう。

> 「rm」を使うといきなり消されてしまうので、それを防止する「trash」のようなプログラムもあります（標準ではインストールされません）。こういった別のコマンドを別途インストールすることでファイルやディレクトリを一旦ゴミ箱に移動することができます。

「rm」コマンドを必ず「-i」オプション付きで実行するためには、コマンドにいちいちオプションを付けずシェルの alias 機能を使ってあらかじめコマンドを登録しても良いでしょう。alias とは任意のコマンドを別名で置き換えることができるものです。alias については後の章で解説しましょう。

複数のファイルやディレクトリをまとめて操作する

複数のファイルやディレクトリをまとめて移動したり消去したい時は、ファイル名やディレクトリ名の代わりに「*（アスタリスク）」を使ってワイルドカード指定をします。Web 制作の作業中に特定の拡張子が含まれているものだけを移動したり消したいことは良くあることですね。以下のように「*.php」とすれば、themes ディレクトリ内の拡張子「.php」を持つファイルがすべて消去されます。

> ワイルドカードを使ったコマンドの実行

```
$ rm ./themes/*.php
```

以下のコマンドを実行すれば「images」ディレクトリ内の「common-」で始まるファイルがすべて、「images/common/」ディレクトリに移動します。

> ワイルドカードを使ったファイルの移動

```
$ mv ./images/common-* ./images/common/
```

ワイルドカードで移動や消去、コピーする対象を指定する場合で中にディレクトリが含まれる時は「-R」や「-f」オプションが必要です。「-f」オプションは「force（強制的に）」という単語の頭文字です。

複数のファイルを結合する

最初の方で紹介した「cat」コマンドを使えば、複数のファイルを結合してひとつのファイルにすることができます。以下のように、結合したい複数のファイル名を「>」で渡してあげると「a.txt」と「b.txt」の内容が結合されて新しく「concat.txt」が生成されます。

> ファイルの結合

```
$ cat a.txt b.txt > concat.txt
```

この「>」を「**リダイレクト**」と呼んでいます。ファイルが既に存在している場合は、内容が消去され（上書きされ）「a.txt」「b.txt」の内容に変わるので注意が必要です。ファイルが既に存在していて、そこに新たに「c.txt」などの内容を追加したい場合は「>>」を使います。

> 既存のファイルに追加

```
$ cat c.txt >> cancat.txt
```

「cat」コマンドを利用すれば、「JavaScript のライブラリなどを使用する際にライセンステキストを付与する」ような作業も、わざわざファイルを開いてコピー&ペーストすることなくコマンドから一発で付け足すことができます。

ファイルやディレクトリの所有者やパーミッションを変更する

UNIX 系の OS では、ファイルやディレクトリには所有者とグループが設定されています。OS X の場合はシステムが利用するものは、所有者が「root」でグループが「wheel（または admin）」になっています。以下のコマンドを実行するとリストの中の項目の所有者とグループが「root:wheel」になっているでしょう。

> '/usr/bin' をリスト

```
$ ls -l /usr/bin
```

一方、デスクトップや自分が作成したディレクトリなどをリストすると「自分のアカウント名:staff」になっているはずです。

> デスクトップをリスト

```
$ ls -l ~/Desktop
```

このように UNIX 系の OS は複数のアカウントで使うことを前提としているため、所有者と所属するグループが厳密に分けられて権限が設定されています。

Web サイト制作の現場では、CMS を設置する時などリモートのサーバにファイルをアップロードして実行ファイルに実行権限を与えたり、ディレクトリのパーミッションを変更することがよくあります。もちろんそれはローカルの OS X での作業中でもたびたび発生することでしょう。そんな時は「chmod（CHange MODe）」や「chown（CHange OWNer）」「chgrp（CHange GRouP）」のコマンドを使います。

ファイルやディレクトリのパーミッションを変更するには「chmod」コマンドを使います。「chmod」コマンドの後にファイルやディレクトリのパーミッションを指定してファイル名やディレクトリ名を指定します。もちろんワイルドカード指定もできますので、任意の拡張子を持つファイルやディレクトリ丸ごとパーミッションを変えるといったことも簡単です。

> ファイルのパーミッションの変更

```
$ chmod 755 example.cgi
```

> ワイルドカードによる変更

```
$ chmod 644 *.php
```

> ワイルドカードによる変更

```
$ chmod 777 ./htdocs/uploads/*
```

任意のファイルやディレクトリの所有者とグループを変更する場合は、それぞれ「chown」コマンド、「chgrp」コマンドを使います。一度に所有者とグループを変えたい場合は、以下のように「所有者: グループ名」と所有者とグループ名を「:（コロン）」で区切って一緒に適用すると簡単です（-R は繰り返し処理オプション）。

※PDF 版では「所有者」と「グループ名」を区切る「:」の後に半角スペースが入ってるように見えますが、この位置の半角スペースは不要です。

> 所有者とグループを一度に変更

```
$ chown -R 所有者: グループ名 ファイル名やディレクトリ名
```

いずれのコマンドにしても所有者が「root」である場合は、実行してもパーミッションエラーになります。その場合は「sudo」コマンドと併せて以下のように実行します（sudo コマンドについては後述）。

> 'sudo' 付きでコマンドを実行

```
$ sudo chown -R 所有者名: グループ名 ファイル名やディレクトリ名
```

パスワードが求められたら、自身の OS X パスワードを入力します。「*」を使って任意のディレクトリ以下のファイルをすべて変更することも可能です。

```
$ sudo chown -R 所有者名: グループ名 /usr/local/bin/*
```

管理者としてコマンドを実行する

OS X ではシステムまわりのコマンドを実行したり、管理者（root）でないと開けないファイルがあります。そのようなディレクトリやファイルを扱う場合は、「sudo」をコマンドの先頭に付けて実行します。「sudo」で実行する場合は、管理者パスワードの入力を求められますので自身の OS X パスワードを入力しましょう。

> 'sudo' 付きでコマンドを実行

```
$ sudo コマンド
```

特に最近の Web 制作では「npm」や「gem」などを使う作業も増えています。公式のインストーラでインストールしたために、コマンドの実行時に常に「sudo」を付けている人も多いのではないでしょうか（sudo なしで実行するインストール方法については 5 章以降で解説します）。指定されたコマンドを実行してもパーミッションエラーが出るようであれば、再度「sudo」を付けて実行してみましょう（これも alias 機能を使えば短くできます）。

sudo コマンドは一度パスワードを入力すると初期設定された時間だけパスワード入力が省略されますが、一定時間が経過したあとは再度パスワード入力が求められます。管理者権限が必要な作業をしばらく続ける場合など、いちいちパスワードを入力するのがわずらわしく感じるでしょう。そのような場合は sudo に「-s」を付けて、一時的に root ユーザーになってしまうこともできます。

> 一時的に root ユーザーに変更

```
$ sudo -s
```

コマンドを実行すれば、以後 root ユーザーとしてすべてのコマンドが実行可能です（あまりお薦めはしませんが）。作業が終わったら「control＋D」キーを押して root ユーザーからログアウトしましょう。

root ユーザーになるコマンドは他にもいくつかありますが、一時的に変わるだけなら上記の「sudo -s」を覚えておけば良いでしょう。一般的には sudo コマンドが実行できるユーザーは限定されます。

覚えておくと便利なコマンド

日常の作業では使う頻度は少ないかもしれませんが、覚えておくと便利なコマンドもあります。

コマンドの場所を確認する

ターミナルで作業をしたり、CLI で実行されるツールなどを使うようになると、実際のコマンドがどこにあるかを調べたいときも出てきます。そのような場合は「`whereis`」「`which`」のコマンドが使えます。たとえば、OS X には標準で PHP が入っていますが、コマンドの場所を確認するには以下のように「`whereis`」または「`which`」にコマンド名を付けて実行します。

> whereis コマンドの実行

```
$ whereis php
```

> which コマンドの実行

```
$ which php
```

コマンドを実行すれば場所を教えてくれます。

> 実行結果

```
/usr/bin/php
```

> 既に Homebrew やその他のツールで PHP が別にインストール済みで、それを使っている場合は「`/usr/local/bin/php`」などになるでしょう。

ファイルやディレクトリを圧縮（アーカイブ）・解凍する

手元のディレクトリをまるごと zip 圧縮したり、アーカイブとして tar.gz といった形式でまとめるには、OS の圧縮プログラムを利用しましょう。リモートのサーバにディレクトリの中身をちまちまと FTP でアップロードしたり、また逆にちまちまとダウンロードするようなことはできれば避けたいものです（予期せぬ事態でネットワークが切断されてやり直しになるかもしれません）。

手元のファイルを zip 圧縮するには「zip」コマンドを使います。

> zip でファイルを圧縮する

```
$ zip 保存するファイル名.zip 圧縮するファイル
```

ここでは保存するファイル名が先になることに注意してください。ディレクトリを丸ごと圧縮する場合は「-r」オプションを付けます。「-r」オプションを付けないと空の zip ファイルの完成です。

> ディレクトリを zip 圧縮する

```
$ zip -r 保存するファイル名.zip 保存するディレクトリ
```

zip コマンドは多様なオプションを備えています。「-1」〜「-9」までで圧縮率を変えたり、「-f」オプションで変更したファイルだけ圧縮するようなことも可能です。

では、今度は zip を解凍してみましょう。

> zip ファイルを解凍する

```
$ unzip 解凍したいファイル
```

ディレクトリに入っているものはそのまま解凍されます。オプションで「-Z」を付けると、.zip の中身を表示することができます。

> zip ファイルの中身を確認する

```
$ unzip -Z 中身をみたい zip ファイル
```

インターネット上で公開されているライブラリやフレームワークなどは「tar.gz」形式（tar 形式のアーカイブ＋gunzip）でまとめられているものがあります。これらのファイルもコマンドラインから元のファイルに戻すことが可能です。

> tar.gz 形式を解凍する

```
$ tar -xvzf 解凍したいファイル.tar.gz
```

「tar」コマンドは、tar 形式のファイルを操作するものです。オプションの指定はいろいろありますが、よくオプションとして使われる「-xvzf」は「eXtract/Verbose/gunZip/Filename」の意味があります。「解凍する/冗長に/gunzip 形式も/次のファイル名を」といったところでしょうか。

逆に tar.gz 形式でアーカイブ＋gunzip 圧縮したい場合は、「x」を「c（Create）」に変えるだけです。

> tar.gz 形式で圧縮する

```
$ tar -cvzf 圧縮するファイル名.tar.gz 圧縮する対象のファイルやディレクトリ
```

これで対象となるファイルやディレクトリなどが、tar.gz 形式でひとかたまりのファイルになります。ここでは「-xvzf」「-cvzf」とハイフン付きにしましたが、「xvzf」「cvzf」でも大丈夫です。Gunzip による圧縮をせず、.tar 形式のアーカイブだけ作りたい場合は「z」オプションを抜きましょう。

> tar 形式のアーカイブの作成

```
$ tar cvf 圧縮するファイル名.tar 圧縮する対象のファイルやディレクトリ
```

このように手元のファイルやリモートのファイルは、インターネット経由で FTP などでのダウンロード／アップロードの前に一度丸ごとまとめてから作業する方がスムーズです。

> リモートサーバへの SSH によるログインが可能なら、ローカルやリモートで圧縮・解凍操作をすることにしておけば、FTP でバラバラと大量のディレクトリやファイルを扱うことはないでしょう。

SSH の鍵の作成

SSH でリモートのサーバにログインしたり GitHub などの Git リポジトリにアクセスするなど、従来の FTP に変わって SSH と（その鍵）を使ってサーバとアクセスすることが増えています。環境によってはパスワード認証は許可されず、SSH の鍵認証などでのログインしか許可されないこともあります。SSH 鍵をローカルのマシンで生成するには、以下のコマンドを実行します（オプションで鍵の種別を指定することもできます）。

> SSH の鍵を作成

```
$ ssh-keygen
```

コマンドをそのまま実行すると下記のメッセージが表示されます。「id_rsa」はデフォルトの鍵の名前です。マシンをアップデートしている場合などは既に作っていることも多いでしょうから、新規で作る場合は重複しないようにどこで使う鍵なのかわかりやすい名前を付けて鍵を作成しましょう。

> 生成中の画面

```
Generating public/private rsa key pair.
Enter file in which to save the key (/Users/username/.ssh/id_rsa): （ここに鍵の保存場\
所を記述）
```

任意で鍵の名前を決めてリターンを入力します（この場合は、keyname）。

> 鍵の保存場所と名前を入力

```
Enter file in which to save the key (/Users/username/.ssh/id_rsa): /Users/us\
ername/.ssh/keyname（リターン）
```

鍵にアクセスするためのパスフレーズの入力が求められますので入力します。パスフレーズを忘れると開くことができなくなるので、頭の中にしっかり覚えておくかどこかにこっそりメモしておきましょう。

これで生成される「keyname」というのが秘密鍵、「keyname.pub」と'.pub' が付いている方が公開鍵です。サードパーティのサービスなどに登録する時は公開鍵を登録します。必要になったらテキストエディタなどで開いて内容をコピーするか、後ほど紹介する「pbcopy」コマンドでコピーして登録しましょう。

SSH は「~/ssh/config」でホスト毎の設定が可能です。

（参考資料）ssh-keygen(1) Mac OS X Manual Page

SSH（SFTP）でリモートのサーバにログインする

SSH での接続が可能なリモートのサーバに遠隔ログインしてマシンを操作できます。ローカルのマシン同様に操作することはもちろん、GUI の FTP クライアントがなくとも SFTP を使ってファイルのアップロードやダウンロードも可能です。SSH でリモートサーバにログインするには、パスワードによる認証や SSH の鍵認証を使います。

> パスワードによるログイン

```
$ ssh username@example.com
```

SSH を使ってログインを試みると、初めてアクセスするホスト（IP）の場合は以下のように接続の処理を続けるか確認がおこなわれます。続ける場合は「yes」を入力しましょう。

> 接続の確認

```
The authenticity of host 'example.com (--.--.--.--)' can't be established.
RSA key fingerprint is --:--:--:--:--:--:--:--.
Are you sure you want to continue connecting (yes/no)?  (yes、または no を入力)
```

yes を入力して進むと「~/.ssh/known_hosts」ファイルにホスト名（または IP）が追加されます。

> known_hosts ファイルへの追加の通知

```
Warning: Permanently added 'example.com' (RSA) to the list of known hosts.
```

パスワード認証の場合は、ここでログインパスワードを聞かれるので入力します。

> IP などが既に登録されているのにマシンが変わった場合など、Warning メッセージが出てホストへのログインができないことがあります。この場合は known_hosts から該当ホスト（IP）を見つけ出して削除しましょう。テキストエディタで該当行を 1 行削除しても構いませんが、「ssh-keygen -R」コマンドを使うと簡単に削除することができます。
>
> **> known_hosts からの削除**
>
> ```
> $ ssh-keygen -R hostname
> ```

SSH の鍵認証を使ったログインは「-i」オプションで SSH の鍵を指定します（後述する ssh-agent を使うと便利です）。

> SSH 鍵の認証によるログイン

```
$ ssh -i .ssh/key-name username@example.com
```

ログイン後はリモートサーバの OS にあわせてコマンドを実行します。ここまで紹介したコマンドは同じように使えるでしょう（Linux OS など UNIX 系の OS の場合）。

SFTP を使ってファイルのアップロードやダウンロードをおこなうなら、以下のように「ssh」を「sftp」に変えればログイン可能です。こちらも同じようにパスワードを聞かれたら教えてあげましょう（鍵認証でのログインも可）。

> パスワードによる SFTP ログイン

```
$ sftp username@example.com
```

SFTP（FTP）でログイン後使えるコマンドは限定的なものになります。リモートのサーバではディレクトリのリストを取得するのは「ls」で同様ですが、それと同時にローカルマシン側を操作するコマンドも使えます。

> SFTP でログイン後のリモートのファイルのリスト表示

```
sftp> ls
```

ローカルマシン側の操作をする場合はコマンドの先頭に「l（小文字の L）」を付け加えましょう。ローカルの作業ディレクトリの一覧をリストするには「lls」となります。

> SFTP でログイン後のローカルのファイルのリスト表示

```
sftp> lls
```

他にも「lcd」でローカルのディレクトリの移動が可能です。このように FTP や SFTP で接続している場合は、ひとつのターミナルの画面で双方を操作できるのです。

ヘルプの表示は「?」を入力してリターン。ログアウトするには「bye」を入力してリターン、もしくは「control＋D」を使いましょう。

> ヘルプの表示

```
sftp> ?
```

以下は、参考までコマンドの一覧を載せておきます（環境によって使えるコマンドは異なります）。

> SFTP でログイン後に使えるコマンドの一覧

```
Available commands:
bye
cd path
chgrp grp path
chmod mode path
chown own path
df [-hi] [path]
exit
get [-Ppr] remote [local]
help
lcd path
lls [ls-options [path]]
lmkdir path
ln [-s] oldpath newpath
lpwd
ls [-1afhlnrSt] [path]
lumask umask
mkdir path
progress
put [-Ppr] local [remote]
pwd
quit
rename oldpath newpath
rm path
rmdir path
symlink oldpath newpath
version
!command
!
?
```

「ssh-agent」を使えば SSH の鍵を登録して簡単にログインできます。登録するコマンドは「ssh-add」です。

> SSH 鍵の登録

```
$ ssh-add 鍵の場所
```

オプションを何も渡さない場合は一時的に登録されますが、「-K」オプション付きで鍵を登録するとキーチェーンに鍵が保存され都度オプションを指定する必要はありません。

> SSH 鍵をキーチェーンに登録

```
$ ssh-add -K 鍵の場所
```

鍵が登録されたらオプションは不要でログイン可能です（sftp も同様）。

> 登録後のログイン

```
$ ssh username@example.com
```

登録済みの鍵は「-l（小文字の L）」を付けるとリストされます（-L にすると内容も含めて表示）。

> 登録済みの SSH 鍵の表示

```
$ ssh-add -l
```

（参考資料）ssh-agent(1) Mac OS X Manual Page（参考資料）ssh-add(1) Mac OS X Manual Page（参考資料）ssh-agent の使い方 - Qiita

シンボリックリンクの作成

OS X は、本来の場所とは異なる場所に「エイリアス」としてショートカットを作ることができます。UNIX 系の OS では「シンボリックリンク」という実体をもったように振る舞うリンクを作成することができます（UNIX でのエイリアスは、まったく別の意味の機能をさします）。

> シンボリックリンクの作成

```
$ ln -s 元のファイルやディレクトリ シンボリックリンクの場所（＋ファイル名）
```

このシンボリックリンクは他の UNIX 系 OS でもよく使うので覚えておきましょう。

（参考資料）知っておくと何かと重宝するシンボリックリンクを Mac で作成する方法

現在の日時を表示する

現在の日時を表示したいときは「date」コマンドを実行しましょう。

> 日時の表示

```
$ date
```

カレンダーを表示する

今月のカレンダーが見たかったら「cal」コマンドで。

> カレンダーの表示

```
$ cal
```

英語の発音を確認する

英単語の発音が知りたい場合は、「say」コマンドが使えます。

> 文章の読み上げ

```
$ say hello world
```

コマンドの実行結果を他のプログラムに渡す

「pbcopy」コマンドを使えば、OS X のクリップボードにテキストなどの内容をコピーして再利用することもできます。

> 内容をクリップボードへコピー

```
$ ls ~/Desktop | pbcopy
```

実行してテキストエディタに貼り付けると、デスクトップのファイル一覧が表れるでしょう。「|」は「**パイプ**」と呼ばれるもので、その区切りの前の処理を区切りの後のコマンドに渡します。pbcopy の詳しい使い方は「man pbcopy」で。

このパイプを覚えておくと長いドキュメントや実行結果などを、他のプログラムに渡してみるといったことができます。以下のようにパイプを使って「/usr/bin」のリスト結果を「less」に渡せば、結果を一度に表示するのではなく段階的に見ていくことができます。

> パイプを使った処理

```
$ ls -la /usr/bin | less
```

こうすれば、「grep」コマンドに内容を引き渡して「passwd」の文字列が含まれるものだけを画面に出すことができます。

> 'passwd' の文字列が含まれるものをリスト

```
$ ls -la /usr/bin | grep passwd
```

このパイプを使えば、冒頭で紹介した「ps -ax」コマンドを使って Web サーバが起動しているかを確認することも可能です。

> httpd の文字列が含まれるプロセスだけを表示

```
$ ps -ax | grep httpd
```

起動していなければ「grep httpd」だけが表示されて終了します。

複数のコマンドを一度に実行するには？

ターミナルの操作に慣れてくると、「複数のコマンドを連続して実行できたら…」という衝動にかられるかもしれません。

一般的なコマンドは一度実行するとその処理が終わるまでは解放されません。処理に時間がかかるものや一連のコマンドを連続して打つような操作をする場合は、一度に複数のコマンドを実行してしまえば楽です。以下のコマンドは「example」ディレクトリを作成して、そこに移動して「index.html」ファイルを作成します。

> コマンドを連続して実行する

```
$ mkdir example && cd example && touch index.html
```

このようにそれぞれのコマンドを「&&」で繋げてあげると、そのコマンドの処理が終わったら次、次のコマンドが終わったらまた次の処理と連続して処理をさせることができます（先に紹介したパイプの機能とは違います。パイプはその実行結果を別のコマンドに引き渡すもの）。

bash では「&&」で処理を続けることができますが、ほかのシェルでは場合によっては「&&」が使えないことがあります。「&&」以外にコマンドの行末をあらわすのに「;（セミコロン）」もあります（&& は処理が正常に終了したら次へ移動、; は関係なく次へ）。サイトなどに記載されていることも多い「&&」ですが、コマンドの区切りは「&&」だけでなく「;」もあわせて覚えておくと良いでしょう。

> ';' を使った例

```
$ mkdir example; cd example; touch index.html
```

> コマンドをバックグラウンドで実行したい場合は、コマンドの最後に「&」を付けると処理をバックグラウンドに回すことができます。

> これ以外にも、OS X で使えるターミナルのコマンドは豊富にあります。一部 OS X でしか動かないものもありますが、その多くは他の UNIX でも同じように使えるでしょう。
>
> （参考資料）An A-Z Index of the Apple OS X command line（参考資料）An A-Z Index of the bash command line for Linux

ターミナルでテキストを編集する

ターミナルでは「Vim（Vi）」や「emacs」「nano」といったテキストエディタを使うことができます。

ローカルではグラフィカル・ユーザー・インターフェイスで提供されるテキストエディタで編集する方が早いでしょうが、OS X に限らずいろいろな環境で共通して使えるエディタの簡単な操作方法だけでも覚えておけば、リモートサーバにログインした状態で設定ファイルを変更するといったことも可能です。

Vim（Vi）

Vim は、もともと Vi として提供されていたエディタの改良版です。起動コマンドは「vim」でも「vi」でも構いません。同じものが起動します。

> Vim の起動

```
$ vi
```

Vim には、「ノーマルモード」「挿入モード」「コマンドモード」「ヴィジュアルモード」というモードの概念があります。何か操作を行うときにそれぞれのモードに切り替える必要があります。これが Vim のちょっと難しい部分です。ターミナルの中で操作するため、文書中のキャレットの移動などもすべてキーボードからおこなうことができます。何か操作に困ったり、画面の意味がわからない時は「esc」キーを入力してみましょう。

Vim の起動直後は「ノーマルモード」になっています。コマンドだけ実行した場合は Vim の初期画面が開きますが、そのままモードを切り替えて入力可能です（ファイルの保存は後半を参照）。新規ファイル名を指定すれば、そのファイル名で画面が開きます。

> 新規ファイル名を指定して起動

```
$ vi 作成するファイル名
```

編集したいファイルがある場合は、vi に続いて半角スペースをあけてファイル名を入力します。

> ファイル名を指定して Vim で開く

```
$ vi 編集したいファイル
```

このノーマルモードの場合は、矢印キーの上下左右、またはキーボードの「h」「j」「k」「l」で文書の中のキャレットを移動することができます（※ キーボードの大文字小文字は区別されるので気を付けましょう。以下大文字のキーは Shift キーを同時に押してください）。OS X Yosemite からは、マウスのスクロール操作でキャレットの上下移動が可能です。

- 左に移動（←、または h）
- 右に移動（→、または l）
- 上に移動（↑、または k）
- 下に移動（↓、または j）

移動してみるとわかりますが、これらのキーを入力すると左右は一文字ずつ、上下は行ごとにキャレットが移動するだけです。これだけでキャレットを移動して編集するのは難儀なので、これ以外に最初の行（gg）や最後の行（G）、行の先頭（0 - ゼロ）や行の最後（$）、単語を飛ばして移動（w や b）といったショートカットがあります。慣れないうちは矢印キーの上下左右の移動からはじめてみましょう。

ノーマルモードで開いた文書を編集したい場合は「挿入モード」に切り替えます。挿入モードへの切り換えは、いくつかのキーを押すことで切り替わります。何のキーを入力するかでキャレットの位置を思い通りの位置に変えてから編集可能です。

- i（キャレットの位置から挿入）
- a（キャレットの次の位置から挿入）
- I（行の先頭テキストに移動して挿入）
- A（行末にキャレットを移動して挿入）
- o（下に空白行を入れて挿入）
- O（上に空白行を入れて挿入）
- s（一文字消して挿入）

慣れないうちは「i」キーを押せば、キャレットの位置で挿入モードに切り替わると覚えておけば良いでしょう。挿入モードの時は、通常のテキストエディタ同様に「delete」キーで文字を消すこともできます（control＋H でも）。挿入モードを解除するには「esc」キーを入力します。挿入モードを解除すればノーマルモードに移行します。

ノーマルモードでは、文字を消したり行を丸ごと削除したりすることもできます。

- x（キャレットの位置から 1 文字消す〜 delete）
- X（キャレットの位置から 1 文字消す〜 back space）
- D（キャレットの位置から行末まで削除）
- dd（その行をまるごと削除）
- 数字+ dd（キャレットの位置から指定した数字ぶんの行をまるごと削除）

ノーマルモードで「dd」を入力すれば、キャレットのある行が 1 行まるごと削除されます。「10dd」のように数字を加えて実行すると、キャレットの位置から 10 行が消去されます。このようにモードを切り替えながら編集をおこなうのが Vim の特徴です。文書内の単語を検索したい、編集が終わって保存したい、Vim を終了したいときなどは「コマンドモード」に移行します。コマンドモードは「/（?）」や「:（コロン）」などを入力して切り替えます。

- /（文字列を検索する）
- ?（文字列を検索する）
- :（コマンドの入力）

文書内の単語を検索する場合は「/」を入力して検索したい文字列を入れてリターンキーを押しましょう。「/」で次を検索、「?」で前を検索します。編集が終わったら「文書を保存する」「名前を付けて保存する」「保存しないで閉じる」といった操作をおこなうでしょう。まずは「:」を入力してコマンドモードに切り替えて以下を入力します。

- w（保存）
- wq（保存して終了）
- q（Vim を終了）
- q!（変更を保存せずに Vim を終了）

新規ファイルを「vi」だけで作った場合は、コマンドモードに切り替えて「:w ファイル名」を実行すればファイル名が付与されて保存されます。

> ファイル名を付けて保存

```
:w ファイル名
```

コマンドモード以外にもノーマルモードで直接保存して終了することができます。

- ZZ（保存して終了）

このように Vim は、ノーマルモード、挿入モード、コマンドモードを適時切り替えて文書を編集するテキストエディタなのです。

この Vim の基本操作を覚えてしまえば、OS X だろうが Linux だろうが、FreeBSD だろうがテキストの編集は可能です。リモートのサーバの設定ファイルを編集するといったことも、いちいちダウンロードして書き換えてアップロードするような面倒くさい作業とはおさらばできるでしょう。

（参考資料）Vim 基本操作まとめ - Archiva

nano

Vim（Vi）はその操作に慣れないうちは、文書の編集操作が非常に怖く感じます。そんな時は「nano」を起動してテキストを編集すると良いでしょう。

> nano の起動

```
$ nano
```

> ファイル名を指定して起動

```
$ nano ファイル名
```

nano を起動するとスクリーンの下にメニューとショートカットが表示されます。Vim のようにノーマルモードや挿入モードといった切り換えはありません。そのままキャレットを移動してファイルを編集し、保存することができます。ターミナルの meta キー（option キー）を有効にしていれば、さまざまなショートカットも使えます。

- control＋G（ヘルプを表示。閉じるのは control＋C）
- control＋C（さまざまな操作をキャンセル）
- control＋O（名前をつけて保存する）
- control＋X（nano を終了する）

- control＋W（文字列を検索する）

これ以外にもキャレットの移動などにショートカットが用意されています。

- control＋A（行の先頭に移動）
- control＋E（行の最後に移動）
- control＋space（単語を順にひとつ飛ばしで後ろに移動）
- meta＋space（単語を順にひとつ飛ばしで前に移動）

設定ファイルなどを編集する場合は、そのままではテキストの折り返しの処理がはいってしまい問題が起きる可能性があります。折り返しを無効化して起動すると意図しないエラーは回避できるでしょう。

> 折り返しを無視して起動

```
$ nano -w ファイル名
```

「-w」オプションはテキストの折り返しを無効化します。これ以外にも閲覧専用で開くなどさまざまなオプションがあります。詳しくは「man nano」コマンドで表示されるマニュアルか、「nano --help」を実行して確認しましょう。

どうでしょうか？ここまで説明してきたように日頃の操作で使うコマンドはさほど多くはありません。コマンドの実行方法やオプションの存在、ディレクトリの移動やファイルの作成、コピー、エディタの簡単な使い方ぐらいを覚えておけば、日常的な作業においては困ることはないでしょう。

制作環境構築の下準備

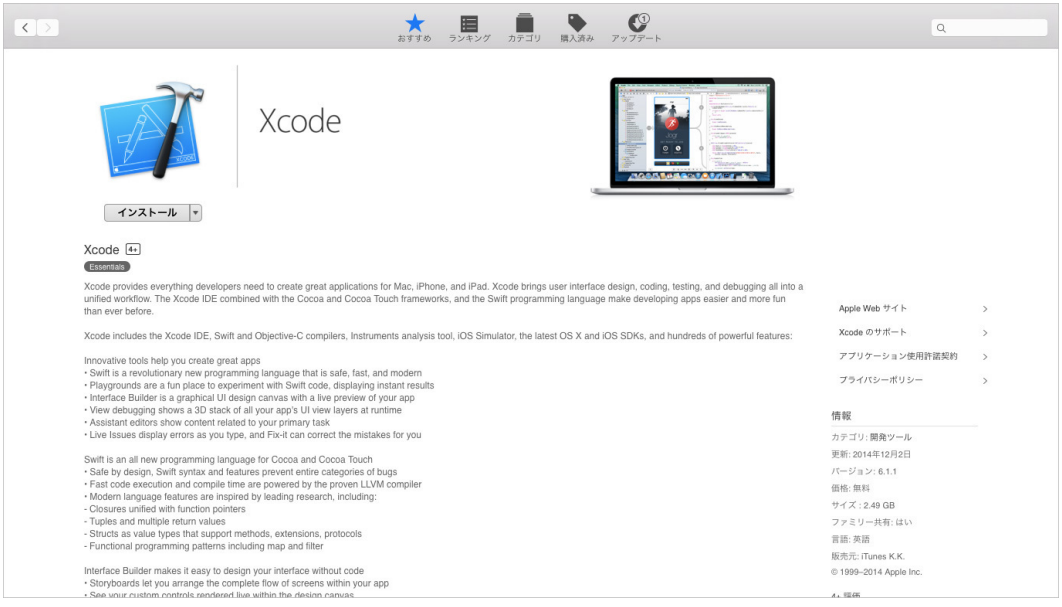
実際の制作環境を作り始める前に下準備をしておきましょう。コマンドラインを使ったツールが増えているいまどきの Web 制作では、それらをすんなりと扱うためにもあらかじめ入れておいた方が良いソフトウェアがあります。本章で紹介する各種ツールはなくてもどうにかなりますが、いずれどこかでこれらのお世話になることになるかもしれません（ないと始まらないこともあります）。ここで前もってインストールだけ済ませておきましょう。

Xcode とコマンドラインツールのインストール

Xcodeは、OS X や iOS のアプリケーションを作成するための統合開発環境です。Web 制作の現場では iOS シミュレータなどで既にお世話になっている方もいらっしゃるかもしませんね。最新の Web 制作ツールを利用するにあたって Xcode 本体は必要ありませんが、Apple の Developer サイトで公開されている「Command Line Tools for Xcode」だけはインストールしておきましょう。

Xcode のダウンロード

Xcode は、App Store からダウンロードしてインストールすることができます。前述のように本体そのものは iOS アプリなどの開発をしない限りは必要ではありません。Web とネイティブで動作するハイブリッドなアプリを開発する時がきたら、「Cordova」のようなツールでコンパイル・シミュレートをする際に必要となるなので入れておくと良いでしょう（ディスク容量が切迫している方は、次の Command Line Tools だけでも構いません）。



Xcode は App Store からダウンロード

Xcode がインストールできているかを確認する場合は、以下のコマンドをターミナルから実行します。

> Xcode の有無を確認

```
$ xcode-select -p
```

既にインストールされている場合は、以下のようにインストール先が表示されます。

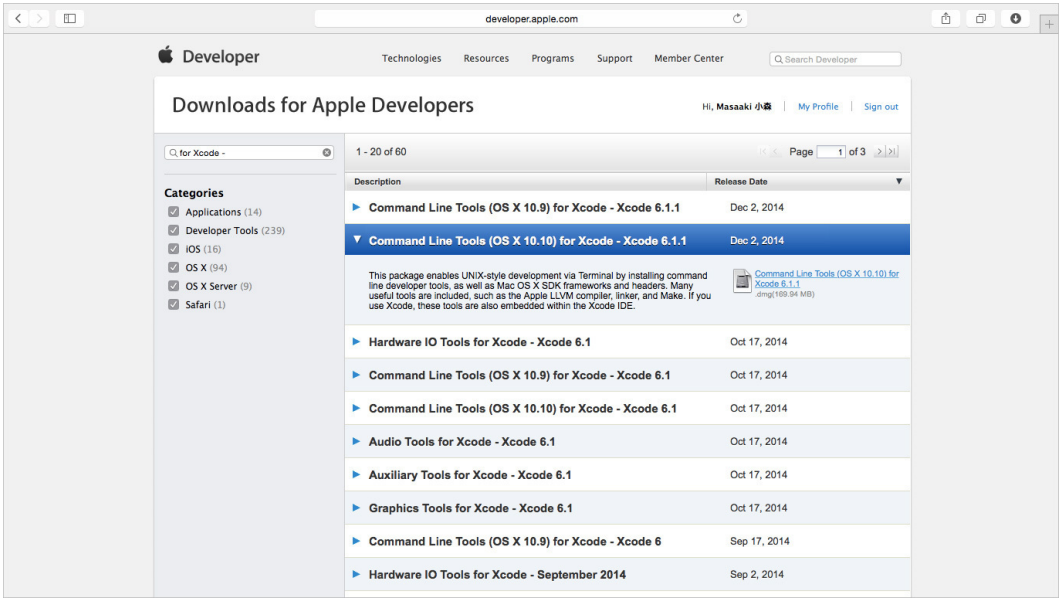
> コマンドの実行結果

```
$ /Applications/Xcode.app/Contents/Developer
```

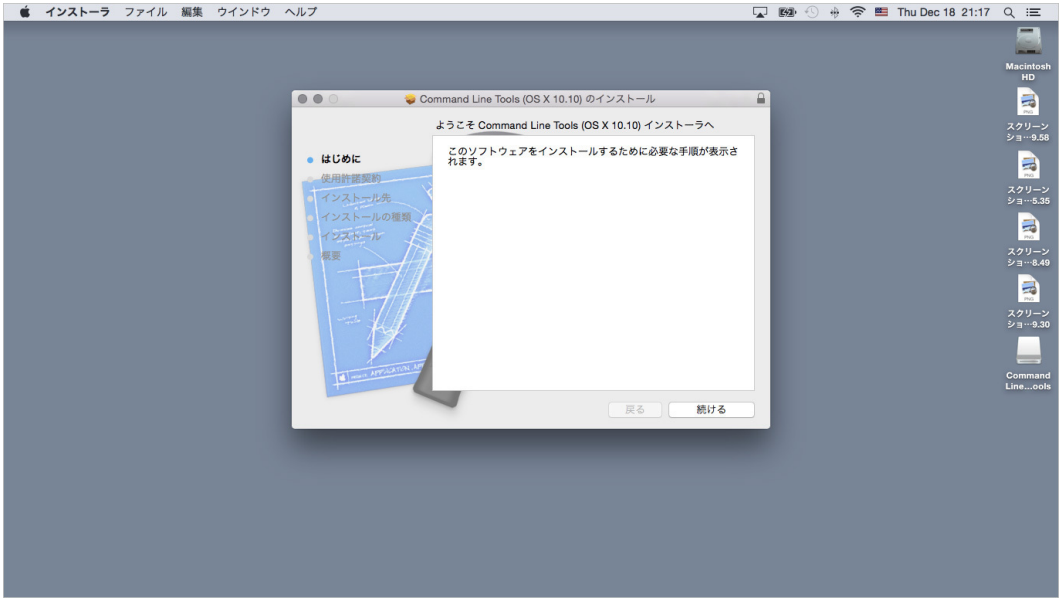
コマンドラインツールのインストール

Apple が提供するソフトウェアのコンパイルに必要なコマンドラインツール一式は、「Command Line Tools for Xcode」として、Xcode とは別に配布されています。このツール群は、Apple の Developer サイトからダウンロードするか（要無償の会員登録）、ターミナルからインストールすることもできます。

Command Line Tools は OS X のバージョン毎に分かれて配布されています。Web サイトからダウンロードする際は、使用中の OS のバージョンにあわせてダウンロードしてください。このファイルは Xcode のアップデートの時などにあわせてバージョンアップされますので、たまには最新版がないか覗いてみると良いでしょう。ダウンロードが終わったらインストールしておきます。



Command Line Tools は OS のバージョン別に配布



インストーラを起動して画面の指示に従ってインストール

既に Command Line Tools がインストール済みかはどうかわからない場合は、以下のコマ

ンドを実行してみると良いでしょう。「gcc」の実行には、Command Line Tools が必要になります。

> インストールされているかの確認

```
$ gcc
```

ダイアログが表示されたらインストールを実行するか、あらかじめターミナルからインストールする場合は以下のコマンドを入力します。

> Command Line Tools をターミナルからインストール

```
$ xcode-select --install
```

インストールが終わったら確認してみましょう。

> gcc のバージョンを確認

```
$ gcc --version
```

インストールされていれば「gcc」コマンドのバージョンが表示されます。

> gcc のバージョン表示

```
gcc --version
Configured with: --prefix=/Applications/Xcode.app/Contents/Developer/usr --w\
ith-gxx-include-dir=/usr/include/c++/4.2.1
Apple LLVM version 6.0 (clang-600.0.56) (based on LLVM 3.5svn)
Target: x86_64-apple-darwin14.0.0
Thread model: posix
```

回線環境をシミュレートする設定のインストール

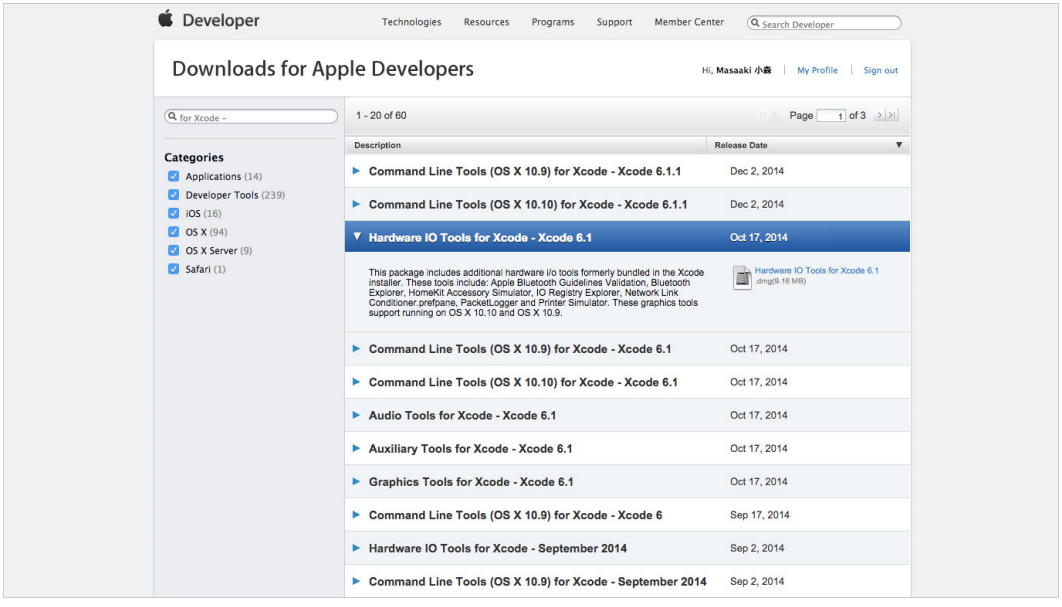
Apple の Developer サイトには、Web 制作の現場で役に立つ「Hardware IO Tools for Xcode」も公開されています。こちらもあわせてダウンロードしておくと便利です。このツール群には「Network Link Conditioner.prefpane」が含まれています。このファイルをダブルクリックしてインストールすると、環境設定のパネルからネットワーク速度に制限をかけることができます。

これがあれば回線環境をシミュレートしながら、制作中もしくは公開中の Web サイトの表示速度チェックが可能です。普段高速な回線で作業していると、遅い回線のことは自分で気にしない限りは気付きませんからインストールしておきたいものです。同様のことは「Charles」のような GUI アプリケーションでも可能です。

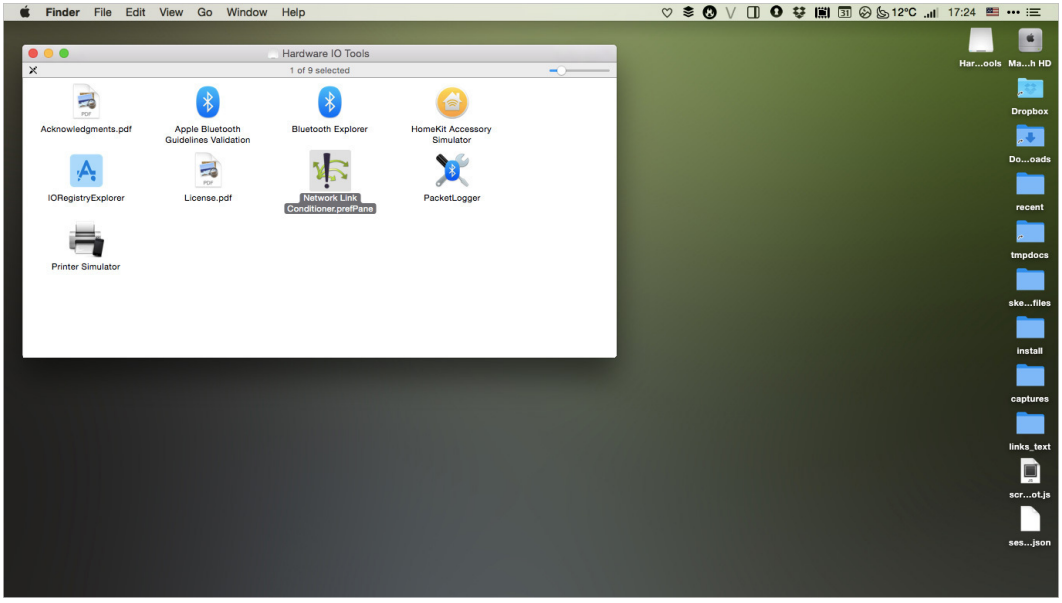
JRE（Java Runtime Environment）のインストール

開発ツールの中には、Java を必要とするものがありますので「JRE（Java Runtime Environment）」をインストールします。Java のパッケージの最新版は Oracle のサイトで公開されていますが、中には Java 6 を要求するものがあり、Apple 社が公開している「Java for OS X 2014-001」が必要なことがあります。あらかじめこちらもインストールしておきましょう。

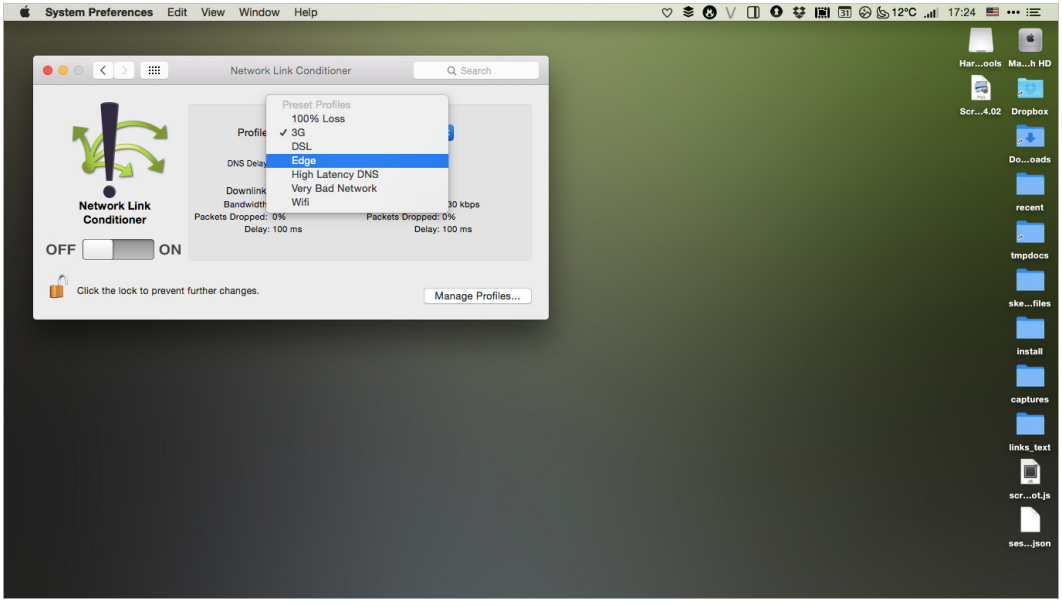
ここまで紹介したもろもろのソフトウェアは、インストールが終わったら特に何もする必要はありません。



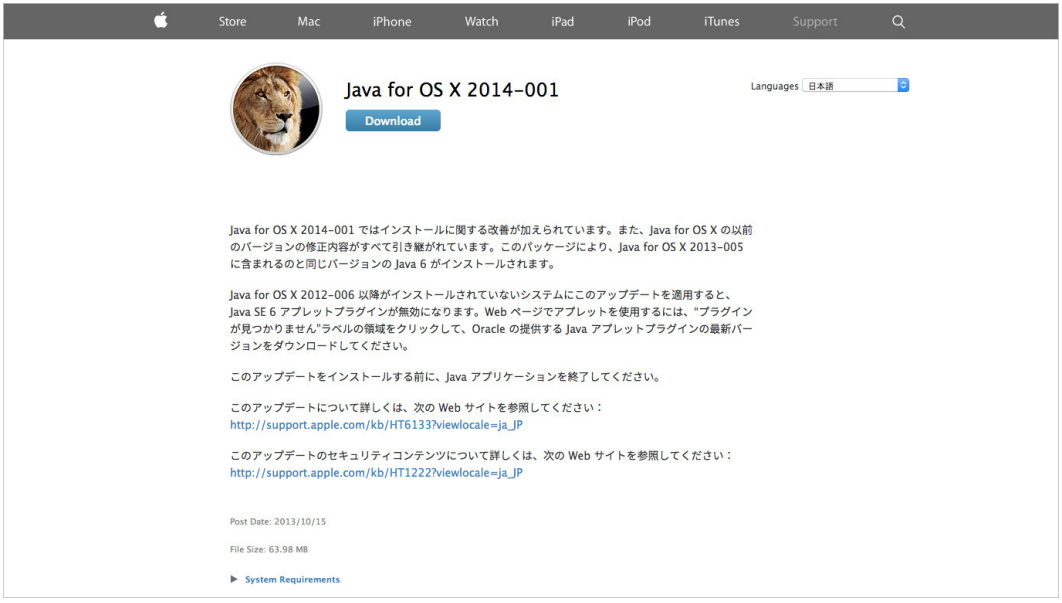
Hardwawre IO Tools も Apple の Developer サイトから



ダウンロードファイルを開いてダブルクリックでインストール



システム環境設定からパネルを開いて通信速度を制限可能



Java for OS X のダウンロード

Homebrew のインストール

OS X は UNIX ベースの OS ですが、UNIX 系の OS でよく利用されるツールがすべて入ってるわけではありません。また、ひとつのバージョンとしてパッケージングされて提供される OS という性質上、システムの出荷時にあらかじめインストール済みの各種ソフトウェアのバージョンがどうしても古くなってしまうことが起こります。「Homebrew」は、そういった問題を解決してくれるパッケージマネージャと呼ばれるソフトウェアです。

Yosemite にインストール済みのソフトウェア

Homebrew をインストールする前に、OS X Yosemite にインストール済みで Web 制作の現場で使いそうなソフトウェアのバージョンを確認してみましょう。各種ソフトウェアのバージョンもコマンドラインから確認できます。

実は OS X には、Web（HTTPD）サーバである Apache、そして PHP もインストールされています。Apache は初期状態でただ起動されてないだけです。Apache のバージョンを確認するには、以下のように「-v」オプションを付けて下記のコマンドを実行します（ソフトウェアによっては、--version オプションで表示）。

> Apache のバージョン確認

```
$ httpd -v
```

コマンドを実行すると以下のように表示されます。

> Apache のバージョン表示結果

```
Server version: Apache/2.4.9 (Unix)
Server built:   Sep  9 2014 14:48:20
```

Apache の執筆時点（2014 年 12 月末）での最新版は「2.4.10」で、これは 2014 年の 7 月にリリースされているものです。PHP は複数のバージョンがありますが、OS X にインストール済みの 5.5.x 系の最新版は「5.5.20」です。このようにちょっとだけ古いバージョンになってしまうのです。以下、OS X Yosemite にインストール済みのソフトウェアのバージョンです。

- Apache: Apache/2.4.9 (Unix)
- PHP: PHP 5.5.14 (cli) (built: Sep 9 2014 19:09:25)

- Ruby: ruby 2.0.0p481 (2014-05-08 revision 45883)
- Python: Python 2.7.6
- Git: git version 1.9.3 (Apple Git-50)

次に執筆時点（2014 年 12 月末）での各ソフトウェアの最新バージョンです。

- Apache: 2.4.10
- PHP: 5.4.36/5.5.20/5.6.4
- Ruby: 2.2.0
- Python: 2.7.9/3.4.2
- Git: 2.2.1

ご覧のようにやはり最新 OS であるとはいえ、バージョンには違いがあるものなのです。

> 決して古いのが悪いというわけではありませんが（重大なセキュリティ上の問題があるものはアップデートされます）、最新の OS であればまだしも数年にわたって使っていくとどうしても古いものを使ってる状態になります。Git については 2014 年 12 月に脆弱性が見つかっていますが、2014 年 12 月末の執筆時点では Xcode の β 版をインストールしないと更新されないようです。

Homebrew とは

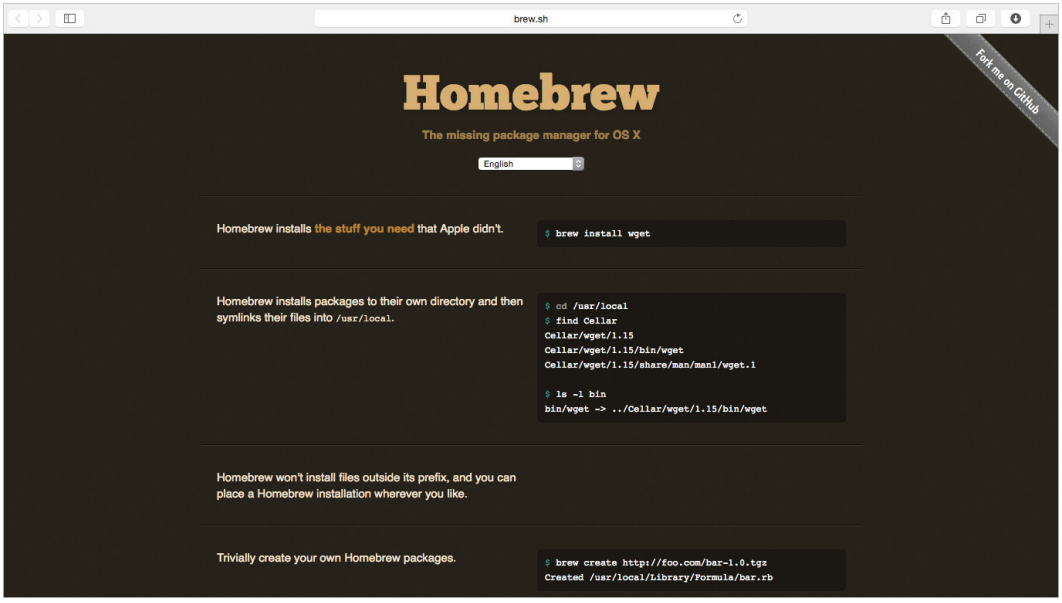
Homebrewは「The missing package manager for OS X」というタグラインが示すように、OS X にはないパッケージを管理するためのソフトウェアです。Linux など UNIX 系の OS に触れる機会がある方には、おなじみのパッケージマネージャーです。

そもそもソフトウェアは、ソースコードがあってそれをコンパイルすることで動作するものです。公開されたソースコード一式をコンパイルするといっても、その他にも必要なソフトウェアが存在します。これらを手動で用意してコンパイルして実行するという作業は、OS に詳しくない人、慣れない人には苦行以外の何ものでもありません。できれば、インストールもアップデートもアンインストールも簡単であるに超したことはありません。

そういった手間を軽減するためにもソフトウェアをすぐに利用できるようにパッケージ化して、その実行に必要な他のソフトウェアとの依存関係までを丸っと管理してしまおうというのがパッケージマネージャーです。各種 Linux 系の OS をはじめとして UNIX 系の OS で

は、使う仕組みこそ異なりますがそれぞれにこのようなパッケージマネージャーが存在しています。この考え方は、いまどきの Web 制作でもよく利用するツールにも採り入れられています。node.js の「npm」や Ruby の「RubyGems（gem）」は、まさにそれ専用のパッケージマネージャーに他なりません。

Homebrew の役割は、「システム内のファイルを直接変更することなく、比較的最新版のソフトウェアを使えるようにする」「インストールされてない UNIX 系のソフトウェアを簡単に使えるようにして、ソフトウェアのバージョンを含めて全体を管理する」ことだと思ってもらえば良いでしょう。

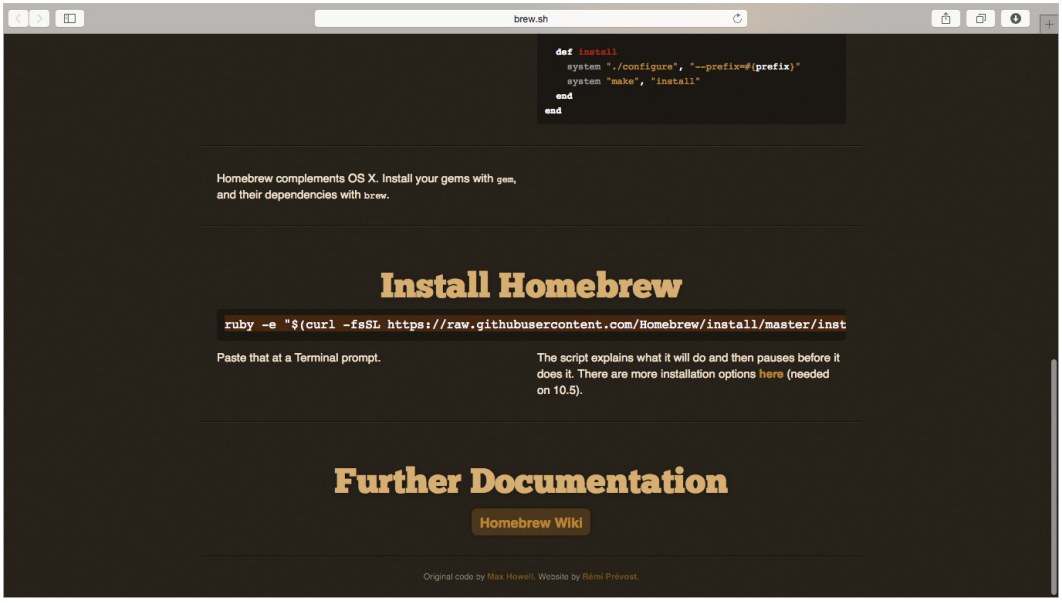


Homebrew の公式サイト

Homebrew のインストール

Homebrew のインストールは簡単です。

公式サイトに記載されたインストールコマンドをターミナルから実行しましょう。下記のコマンドが変わることはないと思いますが、公式サイトのトップページ下にあるコマンドをコピーしてターミナルの画面にペーストして「return キー」を押して実行してください。



Homebrew のインストールコマンドはページの下に

> Homebrew のインストールコマンド

```
$ ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/m\
aster/install)"
```

コマンドを実行すると以下のようにインストールプロセスが始まります。途中パスワードを入力する必要がありますので、ほったらかしにせずしばらく眺めていてください。

```
cipMBA:~ cipher$ ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/
master/install)"
==> This script will install:
/usr/local/bin/brew
/usr/local/Library/...
/usr/local/share/man/man1/brew.1

Press RETURN to continue or any other key to abort
```

コマンドをコピー&ペーストして実行。一度リターンキーを入力する

```
cipMBA:~ cipher$ ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/
master/install)"
==> This script will install:
/usr/local/bin/brew
/usr/local/Library/...
/usr/local/share/man/man1/brew.1

Press RETURN to continue or any other key to abort
==> /usr/bin/sudo /bin/mkdir /usr/local

WARNING: Improper use of the sudo command could lead to data loss
or the deletion of important system files. Please double-check your
typing when using sudo. Type "man sudo" for more information.

To proceed, enter your password, or type Ctrl-C to abort.

Password:
```

パスワードの入力を求められたら入力

> インストール中の出力

```
==> This script will install:
/usr/local/bin/brew
/usr/local/Library/...
/usr/local/share/man/man1/brew.1

Press RETURN to continue or any other key to abort（リターンキーを入力）
==> /usr/bin/sudo /bin/mkdir /usr/local

WARNING: Improper use of the sudo command could lead to data loss
or the deletion of important system files. Please double-check your
typing when using sudo. Type "man sudo" for more information.

To proceed, enter your password, or type Ctrl-C to abort.

Password:（パスワードを入れてリターンキーを入力）

==> /usr/bin/sudo /bin/chmod g+rwx /usr/local
==> /usr/bin/sudo /usr/bin/chgrp admin /usr/local
==> /usr/bin/sudo /bin/mkdir /Library/Caches/Homebrew
==> /usr/bin/sudo /bin/chmod g+rwx /Library/Caches/Homebrew
==> Downloading and installing Homebrew...
remote: Counting objects: 218996, done.
remote: Compressing objects: 100% (57781/57781), done.
remote: Total 218996 (delta 160034), reused 218919 (delta 159981)
Receiving objects: 100% (218996/218996), 48.94 MiB | 4.39 MiB/s, done.
Resolving deltas: 100% (160034/160034), done.
From https://github.com/Homebrew/homebrew
 * [new branch]      master     -> origin/master
HEAD is now at 3bdab72 libmagic: update 5.21 bottle.
==> Installation successful!
==> Next steps
Run `brew doctor` before you install anything
Run `brew help` to get started
```

ここまで表示がでたらインストール完了です。最後に記載されているように、まずは「brew doctor」コマンドを実行しましょう（brew doctor コマンドについては後述）。

> インストール後にセルフチェック

```
$ brew doctor
```

Homebrew のセルフチェックが実行され、問題がなければ以下のような表示が出力されるでしょう。

> 'brew doctor' の実行後

```
Your system is ready to brew.
```

Homebrew の各種ファイルは「/usr/local」ディレクトリ以下にインストールされます。Homebrew を使ってインストールされたソフトウェアは、「/usr/local/Celler」ディレクトリに格納されて、それぞれが「/usr/local/bin」にシンボリックリンクが追加されて使える状態になります。

brew コマンドが実行されない場合は、以下のコマンドを実行してみましょう。

> 環境変数の確認

```
$ echo $PATH
```

表示結果に「/usr/local/bin」があるか確認してください。

> 環境変数の表示結果

```
/usr/local/bin:/usr/bin:/bin:/usr/sbin:/sbin
```

「/usr/local/bin」が含まれていない場合は、以下のコマンドを実行して環境変数のパスに追加します。パスの解説は後述。

> 環境変数にパスを追加

```
$ echo export PATH='/usr/local/bin:$PATH' >> ~/.bash_profile
```

追加した設定を再読み込みします。

> bash の設定を再読み込み

```
$ source ~/.bash_profile
```

もう一度「brew」コマンドが実行できるか確認しましょう（bre まで入力して tab キーで候補に「brew」が見えれば大丈夫です）。

以下、「brew doctor」を実行して起こりがちなエラーとその対策です。

書き込み権限がないと言われる

> Warning: Some directories in /usr/local/share/man aren't writable.
> This can happen if you "sudo make install" software that isn't managed by Homebrew. If a brew tries to add locale information to one of these directories, then the install will fail during the link step. You should probably chown them:

```
/usr/local/share/man/man3
/usr/local/share/man/man5
/usr/local/share/man/man7
```

「/usr/local」ディレクトリ以下に書き込み権限がないと言われて処理が続けられない場合は、「chown」コマンドを使って所有者とグループを変えましょう。所有者は「自分のアカウント名」、グループは「admin」にすれば大丈夫でしょう。

> 所有者とグループを変更する

```
$ sudo chown -R username:admin /usr/local/*
```

node ディレクトリのヘッダファイルが列挙される

> Warning: Unbrewed header files were found in /usr/local/include.
> If you didn't put them there on purpose they could cause problems when building Homebrew formulae, and may need to be deleted.

```
Unexpected header files:
/usr/local/include/node/ares.h
/usr/local/include/node/ares_version.h
/usr/local/include/node/nameser.h
/usr/local/include/node/node.h
・・・以下ファイル名が続く
```

node.js を既にインストールしていてこのエラーが出る人は非常に多いのですが、「/usr/local/include/node」ディレクトリ内にある node.js のヘッダファイルを消しましょう。

> /usr/local/include/node 内のヘッダを消す

```
$ sudo rm -f /usr/local/include/node/*.h
$ sudo rm -f /usr/local/include/node/openssl/*.h
$ sudo rm -f /usr/local/include/node/uv-private/*.h
```

node.js を公式のインストーラからでなくインストールする方法は後の章で紹介します。そうすればこのエラーとは無縁になるでしょう。

パスを通す？環境変数に追加する？

エンジニアさんのブログ記事などを読んでいるとよく「環境変数に追加する」「パスを通す」という表現がされています。プログラミングやバックエンドのことに不慣れだと、これらの言い回しでまず躓くのではないでしょうか？

通常、システムが利用するコマンドは「/bin」や「/usr/bin」といったディレクトリに実行ファイルを置くようになっています。すべてがシステムに関係するソフトウェアであれば良いのですが、そういうものばかりでもありません。時には今回の Homebrew のように標準ではインストールされておらず、後から自分が追加して実行したいソフトウェアも出てくるでしょう。

システムのコマンドがインストールされているディレクトリに自分が追加する実行ファイルを直接入れても良いのですが、それだとシステムのものなのかどうなのかわからなくなり管理もしにくくなります。そこで、一般的にユーザーが追加するソフトウェアは「/usr/local/bin」や自分のホームディレクトリ直下の任意のディレクトリなどに実行ファイルをインストールします。

しかし、システムがチェックするディレクトリは「/etc/paths」に記述されたものだけなので、それではシェルがそのディレクトリを見つけることができません。そこで必要なのが「パスを通す」という作業です。このパスを通すというのは環境変数の「$PATH」に対して自分のシェルから実行可能なディレクトリを登録する・追加することを意味します。一般的にはbash の設定ファイルである「~/.bash_profile」などにその場所を書いておきます。

> .bash_profile を編集

```
$ nano -w ~/.bash_profile
```

追加したいパスをファイルに追記します。

> .bash_profile を編集

```
export PATH=/usr/local/bin:$PATH
```

パスを追加する

ただ追加しただけはすぐには反映されないので、「source ~/.bash_profile」を実行して再読み込みします。

> bash の設定を再読み込み

```
$ source ~/.bash_profile
```

新しくソフトウェアをインストールした際に、パスの追加を促されたらこの作業をおこないましょう。

自分しか使わないマシンなら「/etc/paths」に直接書くという方法もありますが、一般的な方法をお薦めします。パスを追加したのにコマンドが呼び出せない（反映されない）場合は、source コマンドを実行し忘れていないか、パスの順番がおかしくないか、などを確認しましょう。

パスは個別に指定しても良いですし、1 行でまとめても構いませんが、その記述順が関係します。同一コマンドが複数の場所にある場合は、先に見つけて欲しいパスを先に記述します。パスの区切りは「:（コロン）」でディレクトリ最後の「/（スラッシュ）」は不要です。

> 1 行でパスを追加する

```
export PATH=/usr/local/bin:/bin:/sbin:$PATH
```

シェルの設定ファイルは「.bash_profile」以外に「.bashrc」や「.profile」などいくつかあり、自分のホームディレクトリ直下に配置されますが、それぞれのファイルは読み込まれるタイミングが異なります。詳しいことは以下の記事が参考になるでしょう。

（参考記事）bash の便利な機能を使いこなそう (2/2)

Homebrew のアンインストール

「インストールがうまくいかない」「もう一度入れ直したい」ということもあるかと思います。Homebrew をアンインストールする方法は、公式の GitHub のページに説明とアンインストールスクリプトへのリンクがあります（こちらのコメント欄のスクリプトが良いかもしれません）。手動でよければ、以下のディレクトリやファイルを `rm` コマンドで消去しましょう。

> Homebrew がインストールするファイル

```
"/usr/local/.git"
"/usr/local/.gitignore"
"/usr/local/Cellar"
"/usr/local/Library"
"/usr/local/CODEOFCONDUCT.md"
"/usr/local/CONTRIBUTING.md"
"/usr/local/LICENSE.txt"
"/usr/local/README.md"
"/usr/local/SUPPORTERS.md"
"/Library/Caches/Homebrew"
"~/Library/Caches/Homebrew"
"~/Library/Logs/Homebrew"
"/usr/local/bin/brew"
"/usr/local/share/man/man1/brew"
```

> ブログの記事などでは「`/usr/local/bin` ディレクトリを丸ごと消す」といった記載もありますが、そのディレクトリは他のツールでも使うためまるごと消すのはお薦めしません。

> インストール済みのパッケージがある場合は先に「`brew list`」を実行し、インストールし直すパッケージをメモしておきましょう。リストにはパッケージの追加時に依存関係でインストールされたものも含まれます（使ってるツールだけ入れ直せば依存しているものは自動で入ります）。新しく Homebrew でインストールする際は、「何を追加したか」をどこかにメモしておくのがポイントです。

Homebrew によるツールのインストールと管理

Homebrew のインストールが終わったら、さっそく Homebrew を使ってあると便利なソフトウェアをいくつかインストールしつつ、Homebrew の使い方に慣れていきましょう。

tree のインストールと実行

「tree」は、任意のディレクトリ以下に含まれるファイルやディレクトリをツリー形式のフォーマットで書き出せるソフトウェアです。Web 制作の作業をしていると、ディレクトリ構造をツリー上に表したいこともあります（何が何でも Excel というわけでもないでしょう）。

> ～/Music 以下を tree で表示した結果

```
└── iTunes
    ├── Album\ Artwork
    ├── iTunes\ Library\ Extras.itdb
    ├── iTunes\ Library\ Genius.itdb
    ├── iTunes\ Library.itl
    ├── iTunes\ Media
    ├── iTunes\ Music\ Library.xml
    └── sentinel
```

では、さっそく Homebrew でインストールしましょう。Homebrew の公式リポジトリ（登録先）にあるソフトウェアをインストールするには「`brew install`」コマンドを実行します。「`install`」の後には半角スペースを入れて、インストールしたいパッケージ名を入力します（Homebrew の各種コマンドについては後述します）。

> tree のインストール

```
$ brew install tree
```

コマンドを実行するとインストールプロセスが表示されます。

> インストールプロセス

```
==> Downloading http://mama.indstate.edu/users/ice/tree/src/tree-1.7.0.tgz
==> Patching
==> make prefix=/usr/local/Cellar/tree/1.7.0 MANDIR=/usr/local/Cellar/tree/1\
.7.0
/usr/local/Cellar/tree/1.7.0: 7 files, 128K, built in 2 seconds
```

生ビールの絵文字とともにインストールされたパスが表示されたら終了です。さっそくtreeを使ってみましょう。

> tree コマンドで書類ディレクトリを表示

```
$ tree ~/Documents -L 1
```

どうでしょう？「書類」ディレクトリの1階層目がリストされたでしょうか？コマンドに追加している「`-L 1`」は階層のレベルを表します。この場合は、1階層分だけ表示してくれ、ということですね。

次はHomebrewを使ってOS Xのソフトウェアをインストールしてみましょう。

> Hombrewを使ってソフトウェアをインストールすると、終了直前でインストール先のパスやインストール後実行しなければならないコマンドなどが表示されている場合があります。何かソフトウェアをインストールする時は気をつけてください（インストール終了直後の画面はちゃんと見ましょう）。

OS X のソフトウェアを Homebrew でインストール

Homebrewに「Homebrew Cask」を追加すると、`brew cask` コマンドを使ってOS X用のソフトウェアをインストールすることができます。以下のコマンドを実行してHomebrew Caskをインストールしましょう。

> Cask のインストール

```
$ brew install caskroom/cask/brew-cask
```

これで「brew cask」コマンドが利用可能です。brew cask コマンドを使ってインストールされたソフトウェアは「/opt/homebrew-cask/Caskroom」にインストールされます。

では、Homebrew Cask を使ってソフトウェアを追加してみましょう。OS X の標準機能である「Quicklook」は、Finder 上でファイルを選択してスペースキーを押すことでテキストファイルや画像ファイルの中身を見ることができて重宝します。しかし、標準対応したものしか閲覧できないため、Web 制作の現場などで用いられるファイルはアイコンプレビューだけしか出ないのです。

Quicklook 自体はプラグインで拡張できるため、用途に合わせた便利なプラグインが公開されています。これらを追加すれば、Markdown のファイルや README のテキスト、.json や.csv をプレビューする、画像のプレビューと同時にファイルサイズもチェックするといったことが可能になります。

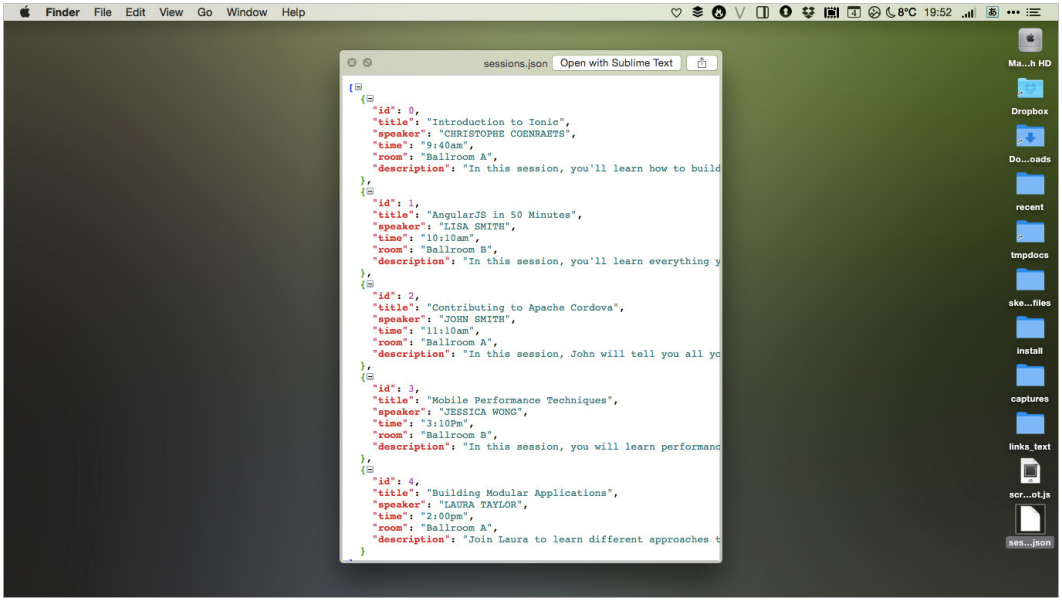
- qlcolorcode: ソースコードのシンタックスカラーリング
- qlstephen: 拡張子のないテキストファイルのプレビュー
- qlmarkdown: Markdown ファイルのプレビュー
- quicklook-json: JSON ファイルのプレビュー
- quicklook-csv: CSV ファイルのプレビュー
- betterzipql: ZIP ファイルの中身をプレビュー
- qlimagesize: 画像ファイルのプレビューにサイズを表示

ここに取りあげたのは一例です。さまざまなプラグインがあるのでインストールしておくと作業効率もあがります。もちろん手動で公開先からダウンロードしてインストールしても良いのですが、Homebrew Cask のリポジトリに登録されているのも多く、Homebrew 経由の方がインストールもその後の管理も簡単になります。

> Cask を使ったインストール

```
$ brew cask install qlcolorcode qlstephen qlmarkdown quicklook-json quickloo\
k-csv betterzipql qlimagesize
```

複数のソフトウェアを一度にインストールするには、各ソフトウェア名を半角スペースで区切ります。コマンドは通常の `brew` コマンドと変わりません。`brew install` が `brew cask install` になる感じです。インストールが終わったら、これまではアイコンしか表示されなかったファイルを Quicklook してみましょう。



JSON ファイルもご覧の通り QuickLook でプレビュー可

インストール済みの Cask を表示するには以下のコマンドを実行します。

> Cask 経由でインストール済みのソフトウェアの表示

```
$ brew cask list
```

> コマンドの実行結果

```
betterzipql qlimagesize qlstephen quicklook-json qlcolorcode qlmarkdown quic\
klook-csv
```

インストールされた Cask の実体は前述のように「`/opt/homebrew-cask/Caskroom`」にあります。`ls` コマンドを実行すると「`~/Library/QuickLook`」ディレクトリへシンボリックリンクが貼られた状態になっているのが確認できるでしょう。

> ユーザーディレクトリの QuickLook を表示

```
$ ls -la ~/Library/QuickLook/
```

「brew cask help」を実行するとコマンド一覧が表示されます。「man brew-cask」コマンドで詳細なマニュアルを表示します（Q キーで終了）。

公式リポジトリ以外からソフトウェアをインストール

Homebrew は初期設定では公式のリポジトリに登録したソフトウェアしかインストールできません。先ほどの Homebrew Cask は、OS X 用のソフトウェアを登録した別のリポジトリでしたが、同様に UNIX 系のソフトウェアでも公式リポジトリには登録されていないものがあります。

Chapter 02 の「ターミナルの操作に慣れよう」で最後に紹介した「nano」は OS X には標準インストールされていますが、これは Homebrew を使って最新版に置き換えたくても公式リポジトリには登録されていません。このようなソフトウェアは公開先のリポジトリを追加することでインストール可能です。

> nano のインストール

```
$ brew install nano
```

インストールコマンドを実行してもエラーメッセージが表示されます。

> エラーメッセージ

```
Error: No available formula for nano
Searching taps...
homebrew/dupes/nano
```

taps は別のリポジトリを表します。Homebrew が別のリポジトリを検索した結果、「homebrew/dupes/nano」にあるということを知らせてくれます。nano は「homebrew/dupes」にあるようなので「brew tap」コマンドでリポジトリを追加しましょう（追加されているリポジトリを除去する場合は「brew untap」）。

> dupes リポジトリの追加

```
$ brew tap homebrew/dupes
```

リポジトリをアップデートしてインストールしましょう。

> リポジトリの情報のアップデート

```
$ brew update
```

> nano のインストール

```
$ brew install nano
```

依存するソフトウェアとともにコンパイルされてインストールが実行されます。

> インストールプロセス

```
==> Installing nano from homebrew/homebrew-dupes
==> Downloading http://www.nano-editor.org/dist/v2.2/nano-2.2.6.tar.gz
==> Patching
==> ./configure --prefix=/usr/local/Cellar/nano/2.2.6 --sysconfdir=/usr/loca\
l/et
==> make install
/usr/local/Cellar/nano/2.2.6: 43 files, 592K, built in 15 seconds
```

これで nano も最新版が使えるようになりました。バージョンを確認します。

> nano のバージョンを確認

```
$ nano --version
```

> Homebrew でインストールされた nano

```
GNU nano version 2.2.6 (compiled 23:35:46, Dec 31 2014)
```

インストールされた nano の実行ファイルは「/usr/local/bin/nano」で、nano コマンドを実行するとこちらが起動します。OS X にあらかじめ入っている nano は「/usr/bin/nano」に残ったままです（つまり、消去も上書きもされていません）。あくまでも環境変数のパスの優先順位で「/usr/local/bin」が「/usr/bin」より前にあるために、Homebrew の方が使われることになるのです。

> OS 標準の nano のバージョンを確認

```
$ /usr/bin/nano --version
```

> OS 標準の nano

```
GNU nano version 2.0.6 (compiled 16:25:25, Sep  9 2014)
```

「brew uninstall nano」を実行すれば、Homebrew の nano がアンインストールされて再び OS 標準の nano が実行されます。

インストール済みのソフトウェアのアップデート

Homebrew でインストールしたソフトウェアは、リポジトリデータを更新してアップグレード処理を実行しない限りはアップデートされません。「brew update」コマンドでリポジトリデータの更新、必要に応じて「brew upgrade」コマンドを実行してパッケージの最新版をインストールしましょう。

> リポジトリの更新

```
$ brew update
```

```
Last login: Tue Jan  6 11:10:22 on ttys000
cipMBA:~ cipher$ brew update
Updated Homebrew from 37b41a80 to 8c04da8d.
==> New Formulae
atlassian-bamboo  id3ed             nave              unarj
dnscrypt-wrapper  julius            speech-tools      wellington
fabric            libgit2-glib      src
==> Updated Formulae
a2ps                            gtk-doc                         pulse
abcde                           gwenhywfar                      pwgen
amtterm                         heroku-toolbelt                 pyenv
android-sdk                     homebrew/dupes/less             pyenv-ccache
ant                             homebrew/dupes/whois            pyenv-pip-rehash
antlr                           html-xml-utils                  pyenv-virtualenv
apgdiff                         htmlcleaner                     pyenv-virtualenvwrapper
ats2-postiats                   htop-osx                        pyenv-which-ext
audiofile                       icoutils                        pypy
avidemux                        ipe                             pypy3
aws-elasticbeanstalk            jenkins                         python
bash-git-prompt                 jpeg                            python3
bashdb                          jpegoptim                       qemu
bashish                         ktoblzcheck                     quilt
bcrypt                          lame                            readline
bibutils                        libgetdata                      recutils
bitchx                          libgphoto2                      renameutils
blueutil                        libsass                         rethinkdb
```

brew update コマンドを実行

「`brew update`」を実行するとリポジトリデータが更新されて、新規追加されたパッケージ、更新されたパッケージなどがリストされます。Homebrew は、通常は使わない Formula や Keg、Celler など酒の醸造に関する単語が用いられるため非常に紛らわしいのですが、アップデートされた Formula のリストが出てきたら次の「`brew outdated`」コマンドを実行してみましょう。

> 更新されたパッケージをリスト

```
$ brew outdated
```

```
cipMBA:~ cipher$ brew outdated
brew-cask (0.51.1 < 0.52.0)
freetype (2.5.4 < 2.5.5)
xz (5.0.7 < 5.2.0)
cipMBA:~ cipher$
```

brew outdated コマンドを実行

更新されているものがあればリスト表示されます。すべてアップグレードする場合は「brew upgrade」、パッケージを指定してアップグレードする場合は「brew upgrade package-name」のようにパッケージ名を指定してアップグレードします。

> パッケージのアップグレード

```
$ brew upgrade（必要に応じてパッケージ名を指定）
```

```
cipMBA:~ cipher$ brew outdated
brew-cask (0.51.1 < 0.52.0)
freetype (2.5.4 < 2.5.5)
xz (5.0.7 < 5.2.0)
cipMBA:~ cipher$ brew upgrade
==> Upgrading 3 outdated packages, with result:
brew-cask 0.52.0, freetype 2.5.5, xz 5.2.0
==> Upgrading brew-cask
==> Cloning https://github.com/caskroom/homebrew-cask.git
Updating /Library/Caches/Homebrew/brew-cask--git
==> Checking out tag v0.52.0
==> Patching
 /usr/local/Cellar/brew-cask/0.52.0: 2358 files, 9.3M, built in 52 seconds
==> Upgrading freetype
==> Downloading https://downloads.sf.net/project/machomebrew/Bottles/freetype-2.5.5.yosemi
######################################################################## 100.0%
==> Pouring freetype-2.5.5.yosemite.bottle.tar.gz
 /usr/local/Cellar/freetype/2.5.5: 60 files, 2.6M
==> Upgrading xz
==> Downloading https://downloads.sf.net/project/machomebrew/Bottles/xz-5.2.0.yosemite.bot
######################################################################## 100.0%
==> Pouring xz-5.2.0.yosemite.bottle.tar.gz
 /usr/local/Cellar/xz/5.2.0: 59 files, 1.7M
cipMBA:~ cipher$
```

brew upgrade コマンドを実行

アップグレードが終わったら、古いバージョンのパッケージデータを「`brew cleanup`」コマンドで削除します。

> 古いパッケージデータの削除

```
$ brew cleanup
```

```
==> Upgrading brew-cask
==> Cloning https://github.com/caskroom/homebrew-cask.git
Updating /Library/Caches/Homebrew/brew-cask--git
==> Checking out tag v0.52.0
==> Patching
  /usr/local/Cellar/brew-cask/0.52.0: 2358 files, 9.3M, built in 52 seconds
==> Upgrading freetype
==> Downloading https://downloads.sf.net/project/machomebrew/Bottles/freetype-2.5.5.yosemi
######################################################################## 100.0%
==> Pouring freetype-2.5.5.yosemite.bottle.tar.gz
  /usr/local/Cellar/freetype/2.5.5: 60 files, 2.6M
==> Upgrading xz
==> Downloading https://downloads.sf.net/project/machomebrew/Bottles/xz-5.2.0.yosemite.bot
######################################################################## 100.0%
==> Pouring xz-5.2.0.yosemite.bottle.tar.gz
  /usr/local/Cellar/xz/5.2.0: 59 files, 1.7M
cipMBA:~ cipher$ brew cleanup
Removing: /usr/local/Cellar/brew-cask/0.51.1...
Removing: /usr/local/Cellar/freetype/2.5.4...
Removing: /usr/local/Cellar/xz/5.0.7...
Removing: /Library/Caches/Homebrew/xz-5.0.7.yosemite.bottle.tar.gz...
Removing: /Users/cipher/Library/Logs/Homebrew/abduco...
Removing: /Users/cipher/Library/Logs/Homebrew/fish...
Removing: /Users/cipher/Library/Logs/Homebrew/git-flow...
Removing: /Users/cipher/Library/Logs/Homebrew/ncurses...
cipMBA:~ cipher$
```

brew cleanup コマンドを実行

GUI のソフトウェアのようにアップデートの通知が自動的に届いたりといったことはありません。頻繁にアップデートの確認をする必要はありませんが、何か新しくソフトウェアをインストールする時などは必ず「`brew update`」を実行する癖をつけて、そのついでにでもアップグレードがないか確かめてみると良いでしょう。

覚えておきたい Homebrew のコマンド

ここまで Homebrew の使い方に慣れるためにいくつかのコマンドを実行してきました。インストールしたソフトウェアは比較的頻繁にアップデートされるものも多くあります。パッケージリストの更新コマンドである「`brew update`」は、24 時間実行していないと「`brew doctor`」でメッセージを表示します（気にすることではありませんが）。

以下、Homebrew を使うにあたって最低限覚えておきたいコマンドを紹介します。これらの「`brew`」に続く「`list`」や「`update`」などのコマンドは、後の章で解説する npm や gem、他の Linux 系の OS のパッケージマネージャーでも大体似たようなコマンドで同じ操作をおこなうことができます。この Homebrew の操作で「どういうコマンドだと何が実行されるのか」を覚えておきましょう。より詳細なコマンドのリストは「`brew help`」を参照してください。

インストール済みのパッケージをリスト表示するには「`brew list`」コマンドを実行します。

> インストール済みのパッケージをリスト

```
$ brew list
```

登録済みのリポジトリのパッケージリストを更新するには「brew update」コマンドを実行します。「brew upgrade」や「brew doctor」を実行する前には一度アップデートをかけましょう。

> パッケージリストをアップデート

```
$ brew update
```

「brew outdated」は、インストール済みのパッケージでアップグレード可能なソフトウェアをリストします。特定のソフトウェアのバージョンを固定しておきたい場合など、「brew upgrade」コマンドでまとめてアップグレードをかけたくない場合はコマンドを実行して確認すると良いでしょう。

> バージョンアップしたパッケージの表示

```
$ brew outdated
```

インストール済みのパッケージを最新版にアップグレードするコマンドは「brew upgrade」です。特定のパッケージのみアップグレードしたい場合は、「brew upgrade」に続けてパッケージ名を指定します。

> パッケージのアップグレード

```
$ brew upgrade
```

> 特定のパッケージをアップデート（パッケージ名を指定）

```
$ brew upgrade package-name
```

インストールしたいパッケージ名がわからない時は、「brew search」コマンドに続けて検索したい文字列を指定します。文字列が含まれているパッケージが結果としてリストされます。

> パッケージの検索

```
$ brew search 検索したい文字列
```

パッケージの情報を見るには「brew info」コマンドを使います。

> パッケージの情報を表示

```
$ brew info package-name
```

不要になったパッケージをアンインストールする場合は「brew uninstall」コマンド、もしくは「brew remove」コマンドに続けてパッケージの名前を入れましょう。

> パッケージのアンインストール

```
$ brew uninstall package-name
```

パッケージをアップグレードした後、不要になった古いバージョンは定期的に削除することをお薦めします。放っておくとソースファイルなどがどんどん溜まっていきディスクの容量を圧迫していくだけです。

> パッケージの古いバージョンを削除

```
$ brew cleanup
```

Homebrew のインストール直後に実行した「brew doctor」は、Homebrew の環境をセルフチェックするコマンドです。エラーメッセージが出たりしてコマンドが実行できない場合は、表示された解決策を試すことで多くの問題は解決します。「brew update」を 24 時間実行していなければ必ず Warning が出ます。一度「brew update」コマンドを実行しましょう。

> Homebrew の環境チェック

```
$ brew doctor
```

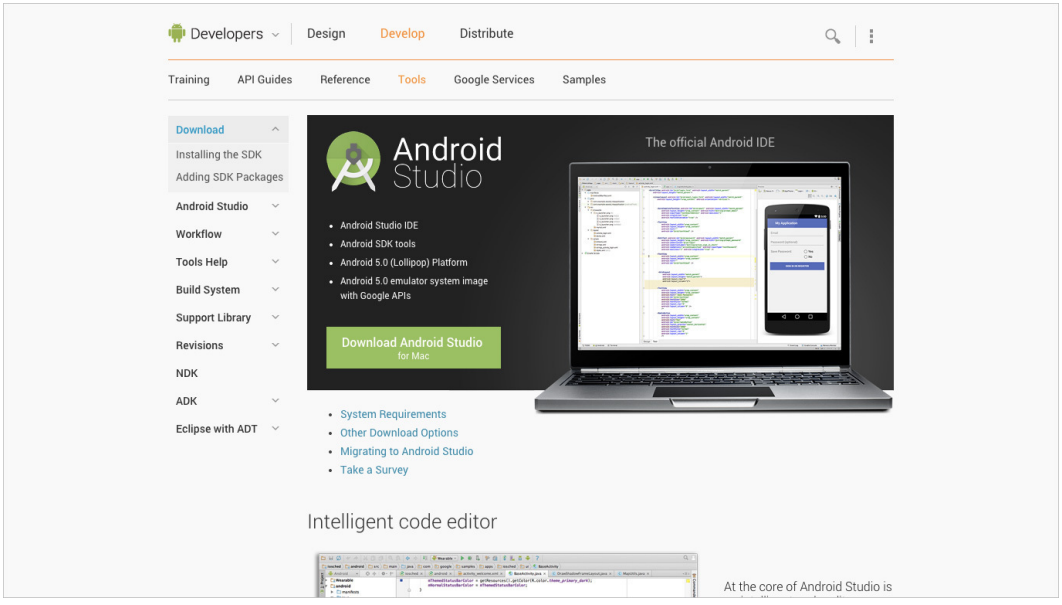
困った時はエラーメッセージをまるごとコピーして、Google や Stack Overflow で検索すると良いでしょう。この際はいずれも英語版のサイト（もしくは結果を英語に絞って）検索した方がすぐ見つかります。

Android SDK Tools のインストール

Android SDK Tools は Android OS を搭載したデバイス向けのソフトウェア開発キットです。最近は「Android Studio」という統合開発環境付きでも配布されています。

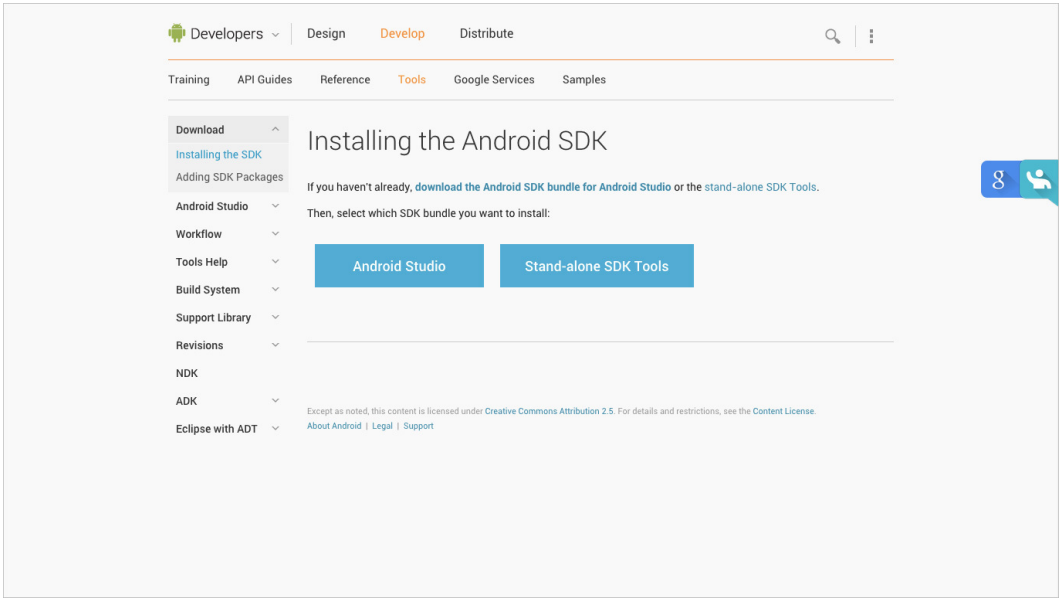
Android SDK Tools のダウンロード

Android SDK Tools はAndroid の公式サイトの「Installing the Android SDK」セクションからダウンロードしましょう。

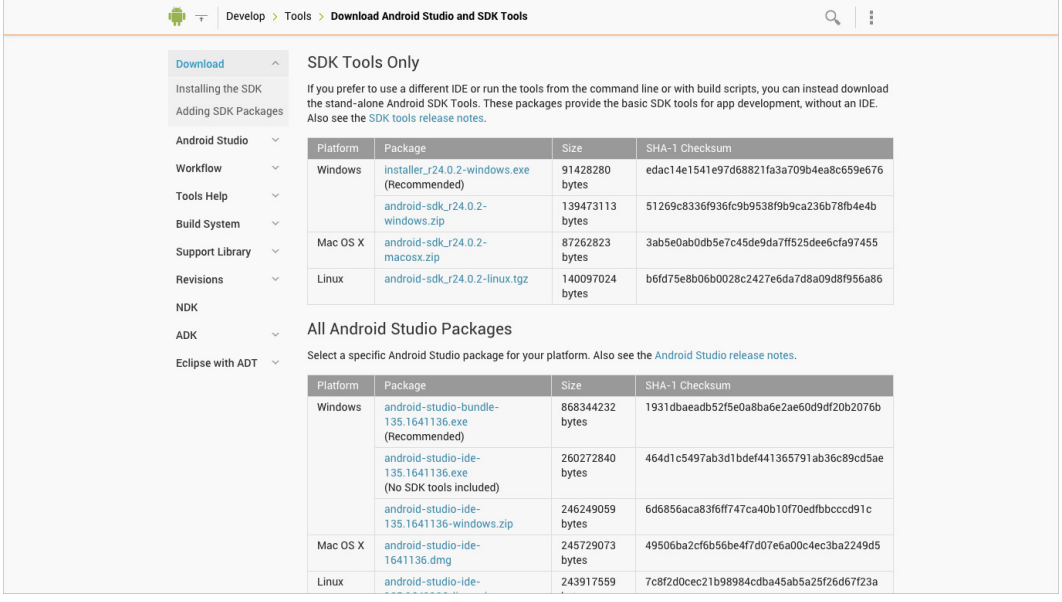


Android Studio は統合開発環境で SDK Tools もセットに

統合開発環境付きの「Android Studio」、もしくは SDK Tools 単体のいずれかでダウンロード可能です。ここでは「Android SDK Tools」を単体でダウンロードしてセットアップしていきます。右側のリンクを進んで「SDK Tools Only」のセクションからファイルをダウンロードしましょう。



右側の Stand-alone SDK Tools のリンクから進む

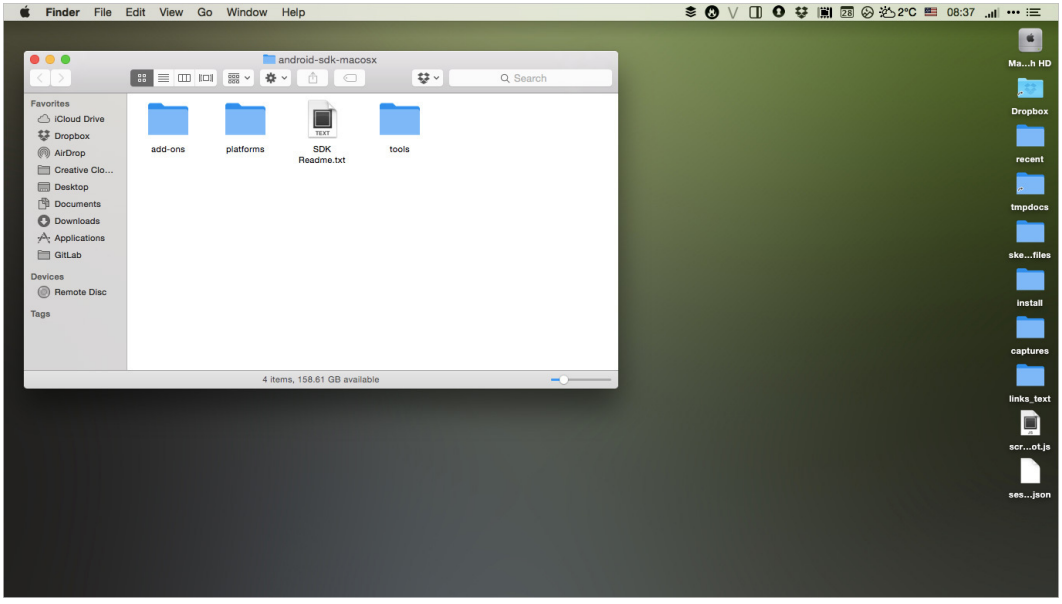


SDK Tools Only のセクションからダウンロード

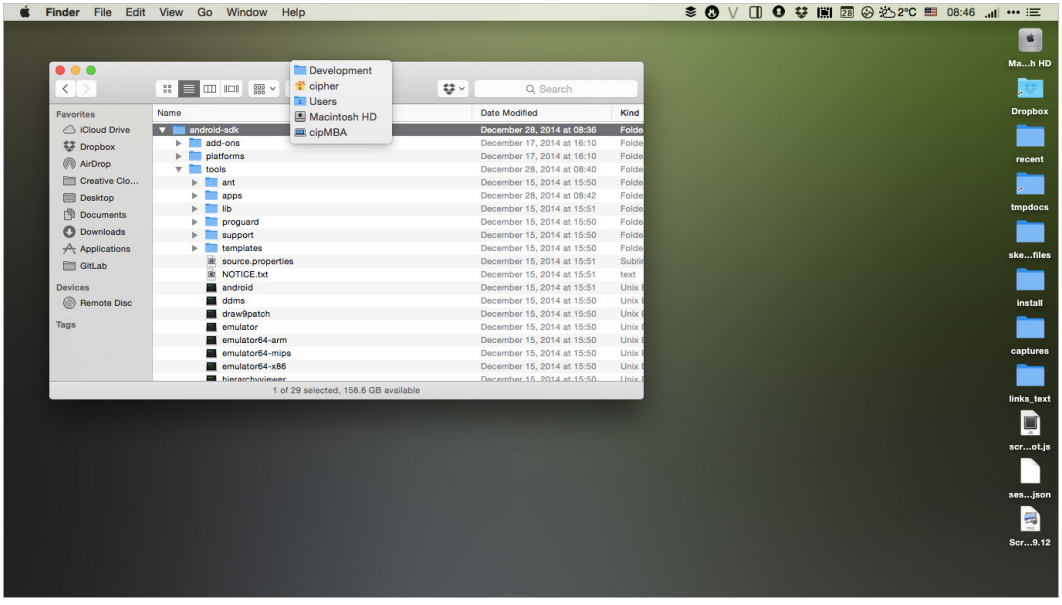
> Android SDK Tools は、セットアップ後 OS のバージョン別のビルドツールやライブラリをインストールすると軽く 10GB を超えますが、Meteor や Ionic のようなフレームワークで Cordova などを使う際に必ず必要になりますのでインストールしておく方が良いかもしれません。

Android SDK Tools のセットアップ

ダウンロードが終わったらファイルを解凍して、ディレクトリを任意の場所に移動します。場所はどこでも構いませんが、自分のホームディレクトリ直下などにする方が良いでしょう。ディレクトリ名は短く変更しても大丈夫です。ここでは「Development」ディレクトリを用意して、その中に「android-sdk」というディレクトリで配置しています。この SDK のディレクトリは、シェルからアクセスできるようにします。



解凍した SDK Tools



自分のホームディレクトリ直下などに移動

移動が終わったら、Vim や nano を使って「`~/.bash_profile`」にパスを追加します。

> `~/.bash_profile` を編集

```
$ nano -w ~/.bash_profile
```

追加するパスは「platform-tools」と「tools」ディレクトリだけで構いません。下記を参考に自分自身のパスに置き換えましょう。パスがわからない場合は、移動した「SDK Tools」のディレクトリをターミナルアプリにドラッグすれば、そのディレクトリに移動できます。「`pwd`」コマンドでパスを表示しましょう。

> ファイルの最後など任意の場所にパスを追加。

```
export PATH="$HOME/Development/android-sdk/platform-tools:$HOME/Development/\
android-sdk/tools:$PATH"
```

パスの追加が終わったら、「`~/.bash_profile`」を再読込します。

> ～/.bash_profile の再読込

```
$ source ~/.bash_profile
```

ターミナルアプリで「android」コマンドへのパスが表示されれば準備は完了です。

> which コマンドによるパスの確認

```
$ which android
```

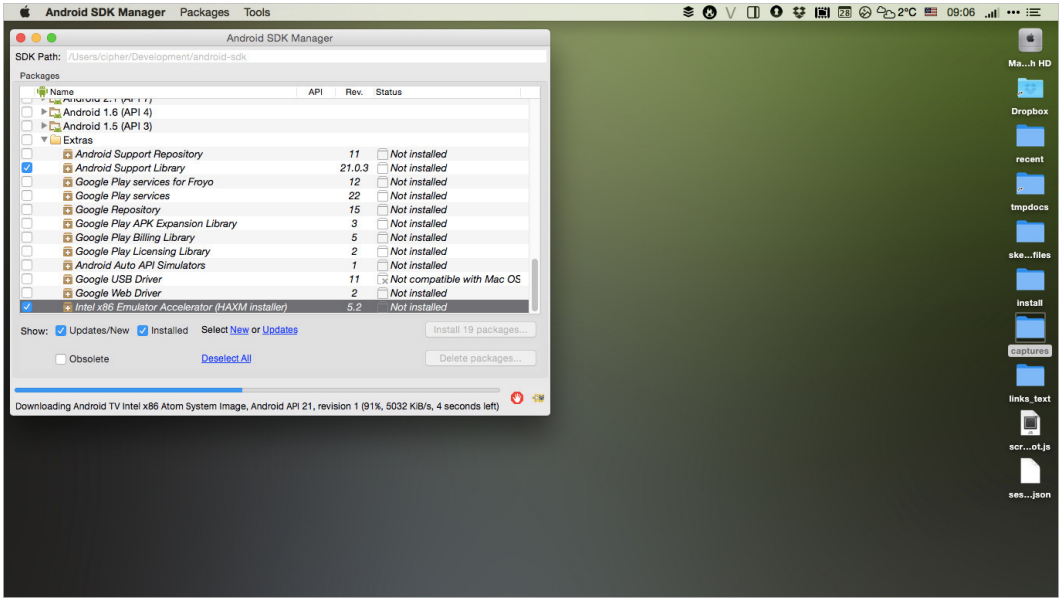
> android コマンドへのパスが表示される

```
/Users/username/Development/android-sdk/tools/android
```

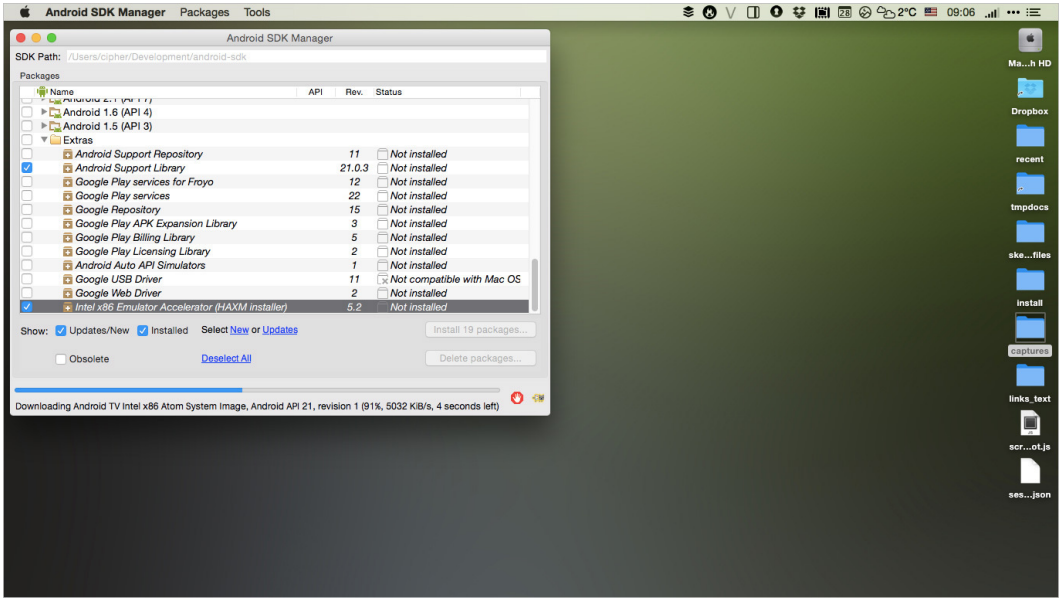
Android SDK Manager の起動

「android」コマンドが使えるようになったら「Android SDK Manager」を起動することができます。起動して Android OS のバージョン別の Platform Tools などをインストールします。これらは必要に応じて追加すれば良いので初期設定のままで大丈夫ですが、ウィンドウの一番下にある「Intel x86 Emulator Accelerator(HAXM Installer)」のチェックを有効にしてインストールを進めます。

> ハイブリッド Web アプリケーションフレームワークの Ionic は、Android 4.4.2（API 19）をターゲットにするのが初期設定のようです。もしこのようなツールを使う場合は必要に応じて追加しましょう。



HAXM Installer のチェックは有効に



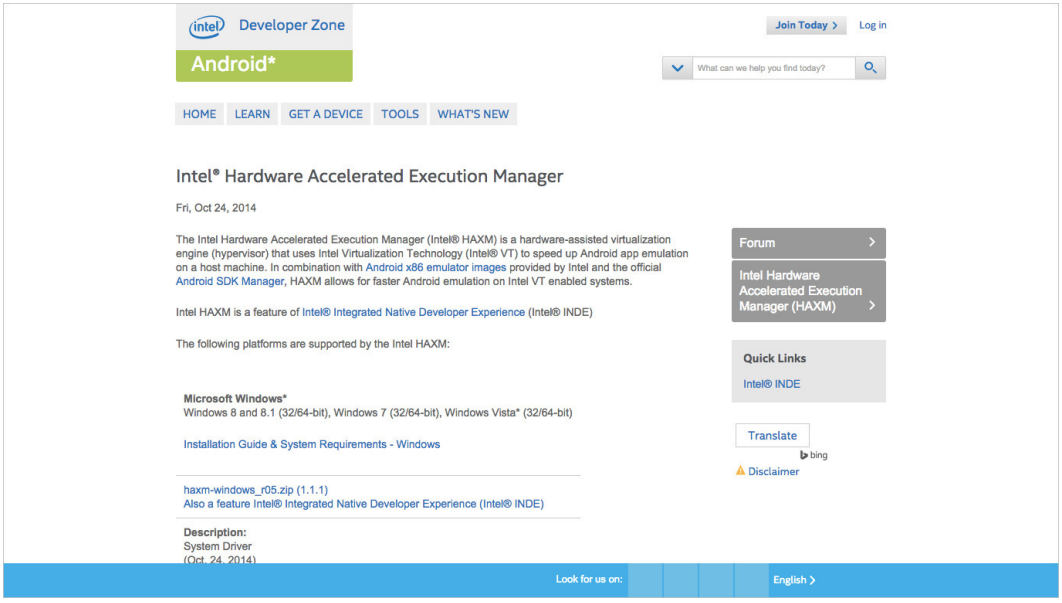
ライセンスアグリーメントに同意してインストール

インストールが終わったら、ウィンドウを閉じて終了します。

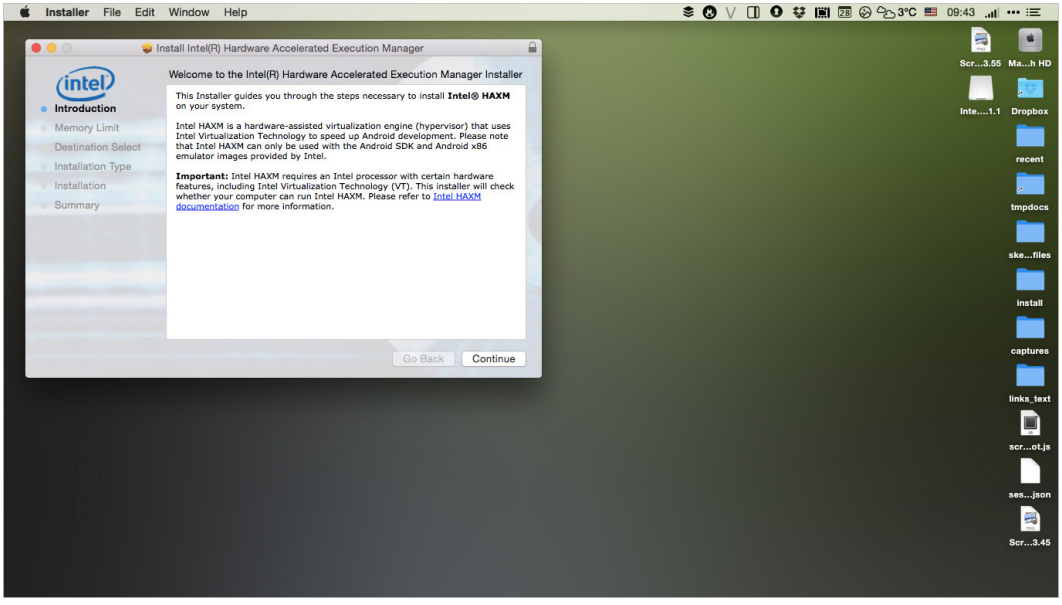
HAXM と Apache Ant のインストール

Android SDK Tools のインストールが終わったら追加でいくつかのソフトウェアをインストールします。

まずは、「Intel® Developer Zone」から「Android* - Intel® Hardware Accelerated Execution Manager」をダウンロードしてインストールします。OS X では、これをインストールし GPU を有効化したデバイスを作らないと起動が遅くとても使い物になりません。ファイルのインストール後は特に何もする必要はありませんが、OS X のメジャーバージョンアップなどを行った際は最新版が配布されていないか公式サイトで確認しましょう。



Intel® のサイトからファイルをダウンロード



ダウンロードしたファイルを開きインストール

またあわせて、デバイスの起動時に必要になる「Apache Ant」もインストールします。こちらは Homebrew を使って下記のコマンドで実行しましょう。

> Homebrew で Ant を追加

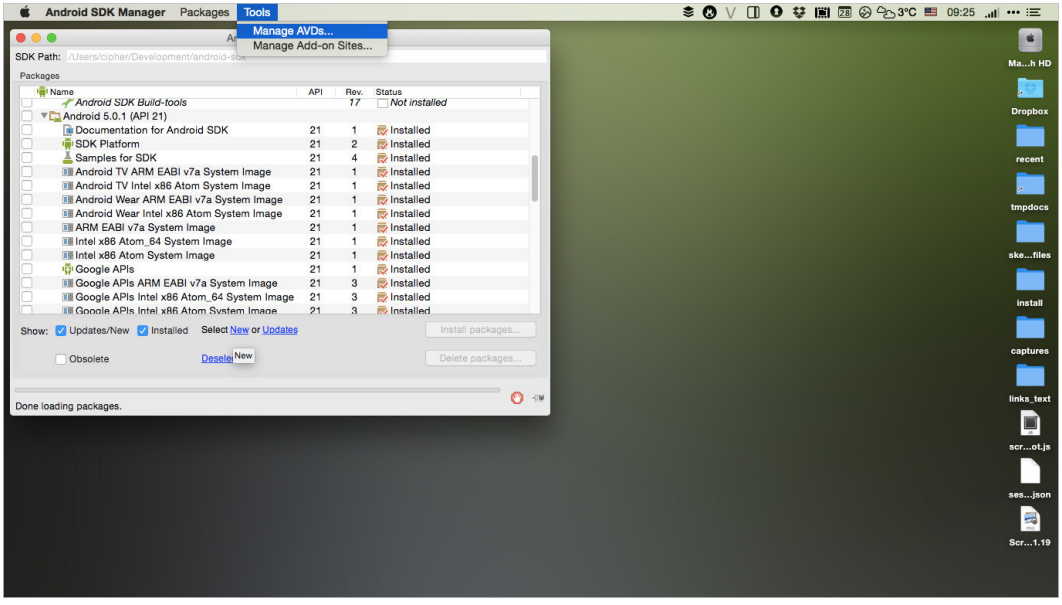
```
$ brew install ant
```

さぁ、次はいよいよデバイスのセットアップです。

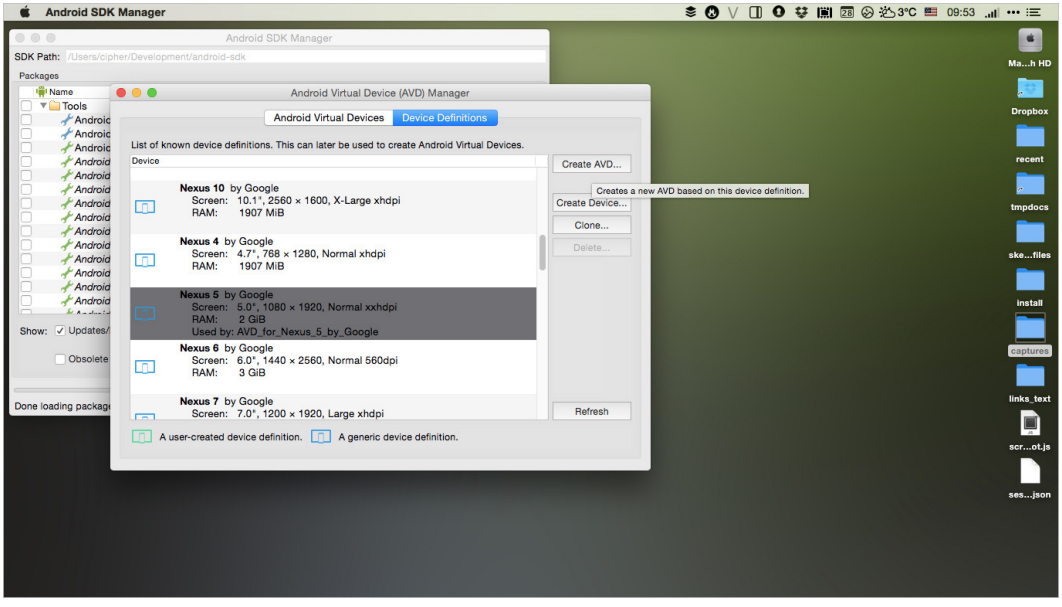
Android デバイスのセットアップ

Android のデバイスをシミュレートするには、あらかじめ仮想デバイスを設定しておく必要があります。再度「android」コマンドを実行し、「Android SDK Manager」を起動します。「Tools → Manage AVDs」メニューを選択して、仮想デバイスを追加します。

「Android Virtual Device(AVD) Manager」が起動するので「Device Definitions」タブをクリックし、作成したデバイスを選択して「Create AVD...」をクリックして設定します。ここでは「Nexus 5」で進めています。



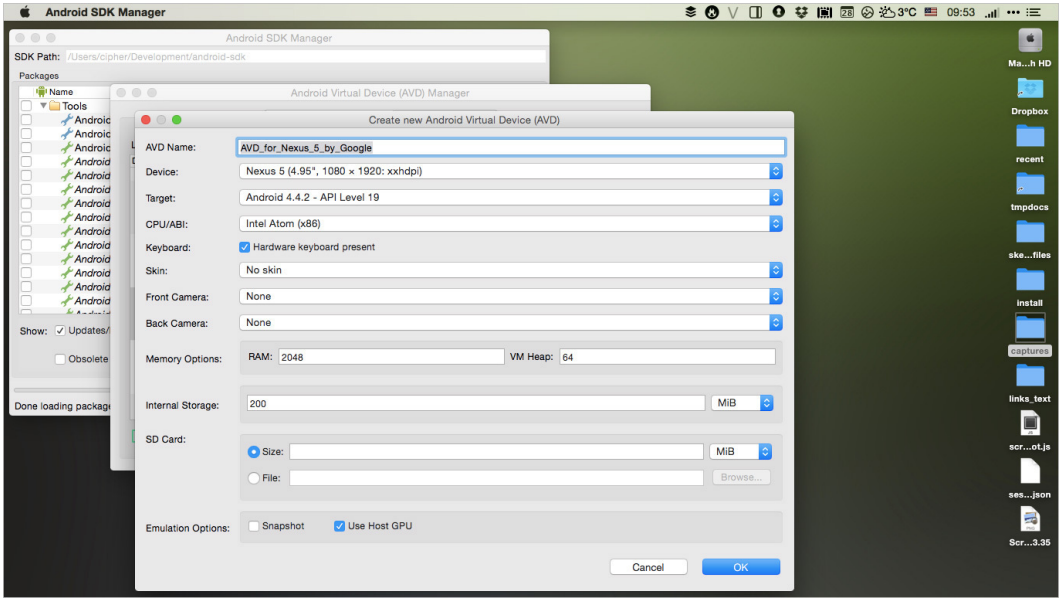
Tools → Manage AVDs を選択



デバイスを選択して「Create AVD...」をクリック

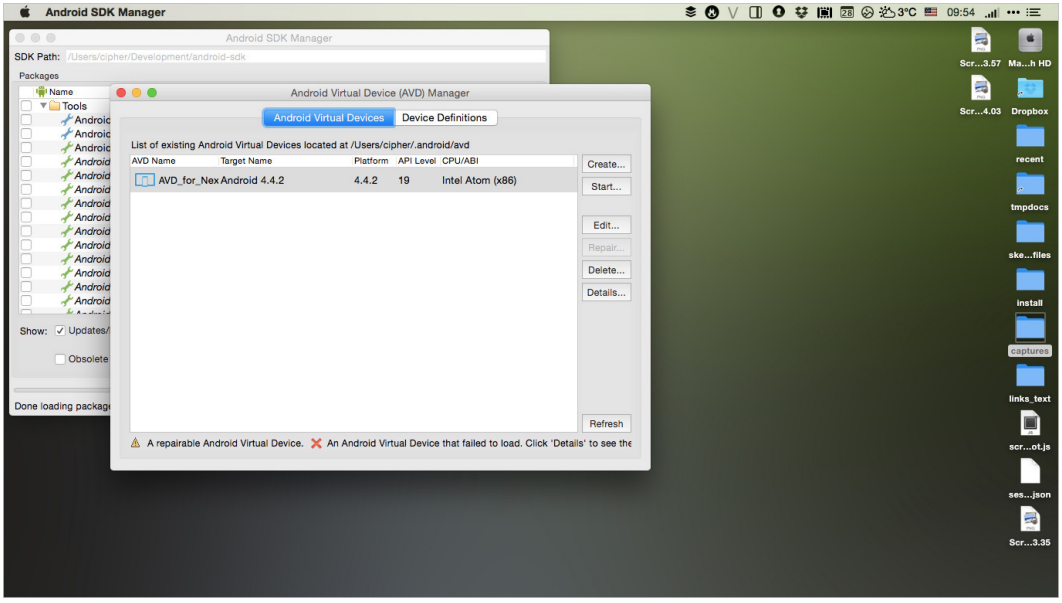
デバイスの設定画面では CPU を「Intel Atom(x86)」にし、Emulation Options の「Use Host

GPU」を有効にするのを忘れないようにしましょう。「OK」ボタンを押して進むと確認ダイアログがでてデバイスが追加されます。



Use Host GPU のチェックを有効に

追加されたデバイスは「Android Virtual Devices」にリストされます。「Edit」で後からデバイスを再編集可能です。「Start...」ボタンをクリックすればこのウィンドウからも仮想デバイスを起動することもできます。



追加された仮想デバイスは後から編集も可能

これで仮想デバイスの追加が終わりました。他のデバイスが必要な時は同じ手順で追加しましょう。