Git 早見表

概要

Git をセットアップする際は、初回のコミット時にユーザ名とメールアドレスが正しく記録されるように、ユーザ名とメールアドレスを設定します。

```
git config --global user.name "<ユーザ名>"
git config --global user.email "<メールアドレス>"
```

Git、GitHub、Heroku について

Git は、小さなものから大規模なものまで、あらゆるプロジェクトを迅速かつ手際よく処理できるように設計された分散型のバージョン管理システムであり、無料かつオープンソースで提供されています。

GitHub は、コード関連の共同作業を行うのに最適なツールです。GitHub では、リポジトリをフォークしてプルリクエストを送信することで、公開および非公開のすべての Git リポジトリを管理できます。

Heroku は、Java、Ruby、Node.js、Clojure など、多数のプログラミング言語をサポートするクラウドアプリケーションプラットフォームであり、Web アプリケーションの開発と展開における新しい手法を提供します。

Git の基本ワークフローの例

新規作成した Git リポジトリを初期化した後、すべてのファイルをディレクトリにステージングし、最後に初期スナップショットをコミットします。

```
$ git init
$ git add .
$ git commit -m 'initial commit'
```

ブランチを新規作成して featureA という名前を付け、チェックアウトしてアクティブにします。次に、ファイルをいくつか編集してステージングし、最後に新規作成したスナップショットをコミットします。

```
$ git branch featureA
$ git checkout featureA
$ (ファイルを編集)
$ git add (ファイル)
$ git commit -m 'featureA を追加'
```

マスターブランチに戻って、先ほど featureA に加えた変更を元に戻し、ファイルをいくつか編集して、新しい変更をマスターブランチのコンテキストで直接コミットします。

```
$ git checkout master
$ (ファイルを編集)
$ git commit -a -m 'ファイルを変更'
```

すべての作業を統合し、featureA の変更をマスターブランチのコンテキストに組み込みます。最後に、featureA ブランチを削除します。

```
$ git merge featureA
$ git branch -d featureA
```

設定と初期化

Git の設定、リポジトリの初期化および複製を行います。

`git config [キー] [値]`	このリポジトリ内の設定値をセットします。
`git config --global [キー] [値]`	このユーザに対するグローバルな設定値をセットします。
`git init`	既存のディレクトリを Git リポジトリとして初期化します。
`git clone [URL]`	URL から Git リポジトリを複製します。
`git help [コマンド]`	指定した Git コマンドのヘルプを表示します。

ステージングとスナップショット

スナップショットおよび Git ステージングエリアを使用します。

`git status`	次回のコミット用にステージングされているファイルと、作業ディレクトリ内の変更されたファイルを表示します。
`git add [ファイル]`	指定したファイルを次回のコミット (ステージング) 対象として追加します。
`git reset [ファイル]`	ファイルのステージングエリアをリセットし、変更内容が次回のコミット時に反映されないようにします (ステージング解除)。
`git diff`	差分 (ステージング前の変更) を取得します。
`git diff --staged`	差分 (ステージング後、コミット前の変更) を取得します。
`git commit`	ステージングされたコンテンツを新しいコミットスナップショットとしてコミットします。
`git rm [ファイル]`	作業ディレクトリからファイルを削除し、ステージングを解除します。
`git gui`	tcl/tk で記述された GUI プログラムを起動し、上述のコマンドをより簡単に使用できるようにします。

ブランチと統合

Git のブランチと一時的な保管機能を使用します。

`git branch`	ブランチを一覧表示します。現在アクティブなブランチの先頭には * (アスタリスク) が表示されます。
`git branch [ブランチ名]`	現在のコミットにブランチを新規作成します。
`git checkout [ブランチ]`	他のブランチに切り替え、そのブランチを作業ディレクトリにチェックアウトします。
`git checkout -b [ブランチ]`	ブランチを作成し、ただちにそのブランチに切り替えます。
`git merge [ブランチ]`	現在アクティブなブランチに別のブランチをマージし、そのマージをコミットとして記録します。
`git log`	コミットのログを表示します。
`git stash`	まだコミットしていない変更内容を作業ディレクトリに一時的に保管します。
`git stash apply`	一時的に保管した変更内容のうち、一番新しいものを再適用します。

共有と更新

別のリポジトリからの更新の取得、マージ、操作を行います。

`git remote add [エイリアス] [URL]`	Git URL をエイリアスとして追加します。
`git fetch [エイリアス]`	リモートの Git からすべてのブランチを取得します。
`git merge [エイリアス]/[ブランチ]`	サーバ上のブランチを現在アクティブなブランチにマージし、最新の状態に更新します。
`git push [エイリアス] [ブランチ]`	リモートの Git リポジトリ上のブランチを更新するために、ブランチで行った変更内容をプッシュします。
`git pull`	現在のブランチで追跡している URL から変更内容を取得し、ただちにマージします。

調査と比較

ログ、差分、オブジェクト情報を詳しく調べます。

`git log`	現在アクティブなブランチのコミット履歴を表示します。
`git log branchB.. branchA`	branchA には含まれているが、branchB には含まれていないコミットを表示します。
`git log --follow [ファイル]`	ファイル変更のコミット履歴を、ファイル名の変更前にまでさかのぼって表示します。
`git diff branchB... branchA`	branchA と branchB との差分 (branchA に含まれ、branchB に含まれない変更点) を表示します。
`git show [SHA ハッシュ値]`	Git 内のオブジェクトを人間が解読できる形式で表示します。
`gitx`	tcl/tk で記述されたプログラムを起動して、コミットログを GUI で表示します。