いちばんやさしい Git&GitHubの教本 人気講師が教える バージョン管理&共有入門

コマンドリファレンス

このコマンドリファレンスでは、『いちばんやさしいGit&GitHubの教本 人気講師が教えるバージョン管理&共有入門』で取り上げたコマンドすべてと、それ以外で知っておくと便利と思われるコマンドの一部を紹介しています。本書で取り上げたものには本書で紹介ラベルが付いており、それ以外には応用ラベルが付いています。応用については、内容が高度なので本書から落としたものですが、いずれもイザというときに役立つものなので、どんなものがあるかだけでも目を通してください。

前半はGitコマンドのパート、後半はそれ以外のコマンドのパートという構成にしており、それぞれ基本的に本書で紹介した順番に並んでいます。

本コマンドリファレンスを読み進めるにあたって、以下の点にご注意ください。

- **コマンド例に使っているファイルやディレクトリのパスは本書で扱ったサンプルに合わせています。実際にお使いの環境やカレントディレクトリに読み替えてください。**
- **パスの指定は絶対パス・相対パスいずれでも可能ですが、コマンドリファレンスでは相対パスでの指定のみ記載しています（絶対パス・相対パスについては書籍P.46を参照）。**
- **ほとんどがWindows前提の実行例となっているため、macOSをご利用の方は、適宜読み替えてください。**

ぜひ、実作業をする際にお役立てください！

gitコマンド

git --version 【バージョンを確認】

Gitのバージョンを確認します。

書式

```
$ git --version
```

本書で紹介 P.52

インストールしているGitのバージョンを確認します。

```
$ git --version
```

git config 【設定の追加、変更、確認】

Gitの設定を追加、変更、確認します。

書式 設定を追加

```
$ git config --global 設定項目名 設定値
```

本書で紹介 P.61

Gitの設定を追加します。すでに同名の設定がある場合は設定値を上書きします。この例では、user.nameの設定項目にichiyasa-gを設定しています。ホームディレクトリの.gitconfigファイルに設定が保存されます。

```
$ git config --global user.name ichiyasa-g
```

書式 ローカルリポジトリ固有の設定を追加

```
$ git config 設定項目名 設定値
```

応用

あるローカルリポジトリ内に移動して--globalオプションを付けずにコマンドを実行すると、そのローカルリポジトリ固有の設定を追加できます。設定値は各ローカルリポジトリの.git/configファイルへ保存されます。
--globalオプションで同じ項目名を設定していた場合も、ローカルリポジトリの設定が優先されます。たとえば、会社で利用しているリモートリポジトリのユーザー名、メールアドレスと、個人用の（GitHubなどの）リモートリポジトリのユーザー名、メールアドレスが異なり、それぞれのローカルリポジトリで設定値を変えたい場合などに使用しましょう。

```
$ git config user.name ichiyasa-g
```

書式 設定値を確認

$ git config 設定項目名

本書で紹介　P.61

指定した設定項目の設定値を確認します。

```
$ git config user.name
```

書式 設定値の一覧を表示

$ git config --list

本書で紹介　P.61

Gitの設定値の一覧を表示します。

```
$ git config --list
```

git init 【ローカルリポジトリを作成】

ローカルリポジトリを作成します。

書式

$ git init

本書で紹介　P.74

ローカルリポジトリを作成したいカレントディレクトリに移動してコマンドを実行します。

```
$ git init
```

git status 【ローカルリポジトリの状態を確認】

ローカルリポジトリの状態を確認します。

書式

```
$ git status
```

本書で紹介 P.74, 141

```
$ git status
```

状況に応じて次のような実行結果が表示されます。

▶ 新規追加されたファイルがある場合

Untracked filesにファイル一覧が表示されます。

```
Untracked files:
  (use "git add <file>..." to include in what will be committed)

        Git_MEMO.md
```

▶ Gitの管理下にあるファイルを変更、削除した場合

not staged（ステージングされていない）ファイルとしてmodified、deletedの状態のファイル一覧が表示されます。

```
Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        deleted:    DELETED_FILE.md
        modified:   MODIFIED_FILE.md
```

▶ ファイルをステージングエリアへ登録した場合

stagedのファイルとして、new fileやmodified、deletedの状態のファイル一覧が表示されます。

```
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        new file:   NEW_FILE.m
        deleted:    DELETED_FILE.md
        modified:   MODIFIED_FILE.md
```

▶変更をすべてコミットした場合

unmodified（変更されていない）状態になります。

```
On branch master
nothing to commit, working tree clean
```

▶使用中のブランチを確認

ファイルの状態にかかわらず、On branchに続いて現在使用中のブランチが表示されます。

```
On branch master
```

git add 【ファイルをステージングエリアへ登録】

指定したファイルや指定したディレクトリ配下のファイルをステージングエリアへ登録します。

書式 ファイルを登録

```
$ git add ファイルパス
```

本書で紹介　P.78, 79

指定したGit_MEMO.mdファイルをステージングエリアへ登録します。

```
$ git add Git_MEMO.md
```

書式 ディレクトリ配下のファイルをすべて登録

```
$ git add ディレクトリパス
```

本書で紹介　P.79

指定したsubDirectoryディレクトリ配下のファイルをすべてステージングエリアへ登録します。

```
$ git add subDirectory
```

書式 ファイルをまとめてステージングエリアへ登録

```
$ git add -A
```

本書で紹介　P.182, 187

ワークツリーにある新規作成したファイルや内容を変更したファイルを、まとめてステージングエリアへ登録します。

```
$ git add -A
```

git diff 【ファイルの差分を確認】

ファイルの修正前と修正後の違い（差分）を確認します。

書式 ワークツリーとステージングエリアの差分を表示

```
$ git diff
```

本書で紹介　P.81, 82

ワークツリーとステージングエリアの差分を表示します。

```
$ git diff
```

次のような実行結果が表示されます。
追加した行は先頭に+が、削除した行には先頭に-が表示されます。

```
diff --git a/Git_MEMO.md b/Git_MEMO.md
index c402133..c3725c3 100644
--- a/Git_MEMO.md
+++ b/Git_MEMO.md
@@ -1,2 +1,5 @@
 # Git学習メモ
-## Gitコマンド
+## Gitコマンド
+
+- ローカルリポジトリを作る
+  - git init
```

書式 ステージングエリアとGitディレクトリの差分を表示

```
$ git diff --cached
```

本書で紹介　P.81,83

ステージングエリアとGitディレクトリの差分を表示します。

```
$ git diff --cached
```

git commit 【ファイルをコミット】

ファイルをコミットします。

書式 ファイルをコミット

```
$ git commit
```

本書で紹介 P.85,86

ステージングエリアに登録済みのファイルをコミットします。コマンドを実行するとテキストエディターが開くので、コミットメッセージを記載します。

```
$ git commit
```

書式 コミットメッセージを指定してコミット

```
$ git commit -m "コミットメッセージ"
```

本書で紹介 P.90

コミットメッセージを指定して素早くコミットします。メッセージは" "で囲んで指定します。

```
$ git commit -m ".gitignoreファイルを追加する"
```

書式 ステージングエリアへの登録とコミットを実行

```
$ git commit -a
```

本書で紹介 P.198

ステージングエリアへの登録とコミットを同時に実行します。
ただし、登録されるのはGit管理下に置かれたファイルの変更のみで、新規作成したようなGitの管理外のファイルは対象外です。

```
$ git commit -a
```

書式 ステージングエリアへの登録とコミットメッセージを指定してコミット

```
$ git commit -am "コミットメッセージ"
```

本書で紹介 P.198

ステージングエリアへの登録とコミットメッセージの指定を同時に行い、コミットします。

```
$ git commit -am ".gitignoreファイルを追加する"
```

書式 直前のコミットを変更し新しいコミットへ置き換え

$ git commit --amend 応用

直前のコミットを変更して、新しいコミットに置き換えます。コミットメッセージの変更や、コミットし忘れた変更を直前のコミットに加えることができます。
実行すると、テキストエディタが開いて直前のコミットのコミットメッセージを変更できます。ステージングエリアに変更を登録しておけば、直前のコミットに含まれて新しいコミットが作られます。
完了するとコミットハッシュが変更されるため、原則としてプッシュすることができなくなります。すでにプッシュしたコミットは変更しないことが望ましいです。

```
$ git commit --amend
```

書式 直前のコミットメッセージを変更

$ git commit --amend -m "コミットメッセージ" 応用

エディタを開かずに、直前のコミットメッセージを素早く変更できます。

```
$ git commit --amend -m "変更後のコミットメッセージ"
```

git checkout 【特定のバージョンをワークツリーへ反映】

ある特定のバージョンをワークツリーへ反映します。

書式 ワークツリーの変更を取り消し

$ git checkout -- ファイル / ディレクトリパス 本書で紹介 P.92,94

ワークツリーの変更を取り消します。

```
$ git checkout -- Git_MEMO.md
```

書式 ブランチを切り替え

```
$ git checkout ブランチ名
```

本書で紹介　P.139, 141, 175

使用するブランチを、指定したブランチに切り替えます。フェッチしたばかりでローカルリポジトリに存在しないブランチを指定した場合、ブランチの作成も行います。

```
$ git checkout update-venue
```

書式 名前を指定して新しいブランチを作成

```
$ git checkout -b 新しいブランチ名
```

本書で紹介　P.182, 183

指定した名前で新しいブランチを作成します。作成元のブランチは、現在使用中のブランチになります。

```
$ git checkout -b speakers-info
```

書式 作成元のブランチを指定して新しいブランチを作成

```
$ git checkout -b 新しいブランチ名 チェックアウト元のブランチ名
```

応用

作成元のブランチを指定して、指定した名前で新しいブランチを作成します。現在使用中のブランチに関わらず、同じブランチが作成元となります。

```
$ git checkout -b speakers-info master
```

git reset　【ローカルリポジトリの状態を戻す】

ローカルリポジトリの状態を、指定した状態へ戻します。

書式 ファイルをステージングエリアからワークツリーへ戻す

```
$ git reset HEAD ファイルパス
```

本書で紹介　P.95, 97

指定したファイルがステージングエリアからワークツリーへ戻ります。HEADとは、最後にコミットした状態を指します。

```
$ git reset HEAD Git_MEMO.md
```

書式 指定したコミットまで取り消し

$ git reset --soft コミット 応用

指定したコミットまでのコミットを取り消します。ワークツリーとステージングエリアは変更されません。コミットハッシュの変更をするので、すでにプッシュしたコミットには実行しないことが望ましいです。たとえば、HEAD~（HEADの一つ前のコミット）を指定して実行すると、HEADを取り消してコミットをHEAD~まで戻します。直前のコミットでGit_MEMO.mdを変更していた場合、結果としてステージングエリアにGit_MEMO.mdの変更が登録されている状態になります。コミットの指定には、ハッシュ値を用いることも可能です。

```
$ git reset --soft HEAD~
```

書式 指定したコミットまで（ステージングエリアへの登録も）取り消し

$ git reset --mixed コミット 応用

指定したコミットまでのコミットを取り消します。ワークツリーは変更されませんが、ステージングエリアへの登録は取り消されます。オプションを指定していない場合も、--mixed オプションと同じ操作になります。コミットハッシュの変更をするので、すでにプッシュしたコミットには実行しないことが望ましいです。
たとえば、HEAD~（HEAD の一つ前のコミット）を指定して実行すると、HEAD を取り消してコミットとステージングエリアを HEAD~ まで戻します。直前のコミットで Git_MEMO.md を変更していた場合、結果としてワークツリーの Git_MEMO.md が変更されている状態になります。コミットの指定には、ハッシュ値を用いることも可能です。

```
$ git reset --mixed HEAD~
```

書式 指定したコミットまで（ワークツリー、ステージングエリアへの登録も）取り消し

$ git reset --hard コミット 応用

コミットを指定して実行すると、そのコミットまでのコミットを取り消します。ワークツリー、ステージングエリアへの登録も取り消されます。コミットハッシュの変更をするので、すでにプッシュしたコミットには実行しないことが望ましいです。
たとえば、HEAD~（HEAD の一つ前のコミット）を指定して実行すると、HEAD を取り消してコミットとステージングエリア、ワークツリーを HEAD~ まで戻します。直前のコミットで Git_MEMO.md を変更していた場合、その変更はコミット、ステージング、ワークツリーから取り消されます。コミットの指定には、ハッシュ値を用いることも可能です。

```
$ git reset --hard HEAD~
```

git rm 【ファイルを削除】

ワークツリーやステージングエリアからファイルを削除します。

書式 ワークツリーからファイルを削除

```
$ git rm ファイルパス
```

本書で紹介 P.98,P101

ワークツリーからファイルを削除し、削除した状態をステージングエリアへ登録します。

```
$ git rm remove_me.txt
```

書式 ワークツリーからディレクトリを削除

```
$ git rm -r ディレクトリパス
```

本書で紹介 P.99

ワークツリーからディレクトリを削除し、削除した状態をステージングエリアへ登録します。

```
$ git rm -r subDirectory
```

書式 ファイルやディレクトリをGitの管理下から外す

```
$ git rm --cached ファイルパス
$ git rm --cached -r ディレクトリパス
```

応用

指定したファイルやディレクトリをGitの管理下から外し、その状態をステージングエリアへ登録します。ワークツリーからは削除されないため、ファイル自体は残ります。

```
$ git rm --cached remove_me.txt
```

または 以下のコマンドを実行します。

```
$ git rm --cached -r subDirectory
```

git log 【コミット履歴を表示】

コミット履歴を表示します。

書式 コミット履歴を新しい順に表示

```
$ git log
```

本書で紹介 P.108, 109

コミット履歴を新しい順に表示します。

```
$ git log
```

書式 差分とコミット履歴を新しい順に表示

$ git log -p

本書で紹介　P.108, 110

差分とコミット履歴を新しい順に表示します。

```
$ git log -p
```

git clone 【リモートリポジトリをコピー】

すでに存在するリポジトリをコピーします。コピーしたリポジトリ用に新しいディレクトリが作成され、Gitディレクトリが格納されます。

書式 リモートリポジトリをコピー

$ git clone リモートリポジトリのURL

本書で紹介　P.127, 129

カレントディレクトリ内にGitリポジトリと同じ名前のディレクトリを作成し、Gitディレクトリを格納します。

```
$ git clone git@github.com:ichiyasa-g/ichiyasaGitSample.git
```

書式 ディレクトリを作成してリモートリポジトリをコピー

$ git clone リモートリポジトリのURL ディレクトリ名

応用

カレントディレクトリ内に指定した名前のディレクトリを作成し、Gitディレクトリを格納します。

```
$ git clone git@github.com:ichiyasa-g/ichiyasaGitSample.git sampleDirectory
```

git remote 【リモートリポジトリの設定を変更、確認】

リモートリポジトリの設定の変更、確認を行います。

書式 リモートリポジトリの設定を確認

$ git remote -v

本書で紹介　P.130

リモートリポジトリの設定を確認します。

```
$ git remote -v
```

次のような実行結果が表示されます。2行表示されているのが、リモートリポジトリとして設定したリポジトリです。

```
origin  git@github.com:ichiyasa-g/ichiyasaGitSample.git (fetch)
origin  git@github.com:ichiyasa-g/ichiyasaGitSample.git (push)
```

書式 指定したURLのリモートリポジトリを設定に追加

$ git remote add リモートリポジトリ名 リモートリポジトリのURL 応用

指定した名前で、指定したURLのリモートリポジトリを設定に追加します。同じリポジトリ内の設定で、同じリモートリポジトリ名を複数回使用することはできません。

```
$ git remote add shiga git@github.com:shiga-iyg/ichiyasaGitSample.git
```

書式 リモートリポジトリの設定を削除

$ git remote remove リモートリポジトリ名 応用

指定した名前のリモートリポジトリの設定を削除します。リモートリポジトリとしての設定が削除されるのみで、そのリモートリポジトリ自体が削除されるわけではありません。

```
$ git remote remove origin
```

書式 リモートリポジトリの名前を変更

$ git remote rename 変更前のリモートリポジトリ名 変更後のリモートリポジトリ名 応用

設定したリモートリポジトリの名前を変更します。

```
$ git remote rename shiga origin
```

git branch 【ブランチの作成、変更、削除、確認】

ブランチの作成、変更、削除、確認を行います。

書式 ブランチを作成

```
$ git branch 新しいブランチ名
```

本書で紹介 P.139, 140

指定した名前のブランチを作成します。ひとつのリポジトリに同じ名前のブランチを複数作成することはできません。

```
$ git branch update-venue
```

書式 ブランチを一覧表示

```
$ git branch
```

本書で紹介 P.140

ブランチを一覧表示します。現在使用中のブランチも表示されます。

```
$ git branch
```

次のような実行結果が表示されます。先頭にアスタリスク(*)が付いているのが、現在使用中のブランチです。

```
* origin
  update-venue
  speakers-info
```

書式 マージ済みのブランチを削除

```
$ git branch --delete ブランチ名
$ git branch -d ブランチ名
```

本書で紹介 P.200

指定したマージ済みのブランチを削除します。マージ済みではない場合、警告が表示され削除に失敗します。なお、現在使用中のブランチを削除したい場合は、実行前に 1 度別のブランチをチェックアウトする必要があります。使用中のブランチは削除することができません。

```
$ git branch --delete update-venue
```

または以下のコマンドを実行します。

```
$ git branch -d update-venue
```

書式 マージ状況にかかわらずブランチを削除

$ git branch -D ブランチ名

本書で紹介 P.200

指定したブランチを削除します。ブランチがマージ済みかどうかにかかわらず、必ず削除に成功します。なお、現在使用中のブランチは削除することができません。使用中のブランチを削除したい場合は、実行前に1度別のブランチをチェックアウトする必要があります。

```
$ git branch -D update-venue
```

書式 ブランチの名前を変更

$ git branch -m 変更前のブランチ名 変更後のブランチ名

応用

作成済みのブランチの名前を変更します。

```
$ git branch -m update-venue update-event-venue
```

git push 【ローカルの内容をリモートリポジトリに反映】

ローカルリポジトリの内容をリモートリポジトリに反映します。

書式 ローカルリポジトリのブランチの内容をリモートリポジトリに反映

$ git push リモートリポジトリ名 ローカルリポジトリのブランチ名

本書で紹介 P.145, 146

ローカルリポジトリのブランチの内容をリモートリポジトリに反映します。同じ名前のブランチがリモートリポジトリに存在する場合はそのブランチが更新され、まだない場合は新たに作成されます。

```
$ git push origin update-venue
```

書式 ローカルリポジトリのブランチの内容をリモートリポジトリの指定したブランチに反映

$ git push リモートリポジトリ名 ローカルリポジトリのブランチ名:リモートリポジトリのブランチ名

応用

ローカルリポジトリのブランチの内容を、リモートリポジトリの指定したブランチに反映します。指定したブランチがリモートリポジトリに存在する場合はそのブランチが更新され、まだない場合は新たに作成されます。

```
$ git push origin update-venue:update-event-venue
```

書式 リモートリポジトリのブランチを削除

$ git push --delete リモートリポジトリ名 ローカルリポジトリのブランチ名 本書で紹介 P.200

$ git push リモートリポジトリ名 :ローカルリポジトリのブランチ名

指定したリモートリポジトリのブランチを削除します。

```
$ git push --delete origin update-venue
```

または以下のコマンドを実行します。

```
$ git push origin :update-venue
```

git pull 【リモートリポジトリの内容をワークツリーへ反映】

リモートリポジトリの内容をローカルリポジトリに取得し、ワークツリーに反映します。

書式 リモートリポジトリのブランチの内容を現在使っているブランチに反映

$ git pull リモートリポジトリ名 リモートリポジトリのブランチ 本書で紹介 P.173, 174

指定したリモートリポジトリのブランチの内容を現在使っているブランチに反映します。

```
$ git pull origin update-venue
```

書式 リモートリポジトリのブランチの内容を指定したローカルリポジトリのブランチに反映

$ git pull リモートリポジトリ名 リモートリポジトリのブランチ名:ローカルリポジトリのブランチ名 応用

指定したリモートリポジトリのブランチの内容を、指定したローカルリポジトリのブランチに反映します。ローカルリポジトリにそのブランチがまだ存在しない場合、新たに作成されます。

```
$ git push origin update-event-venue:update-venue
```

git fetch 【リモートの内容をローカルリポジトリに反映】

リモートリポジトリの内容をローカルリポジトリに反映します。ワークツリーは変更しません。

書式

```
$ git fetch リモートリポジトリ名
```

本書で紹介 P.173, 175

指定したリモートリポジトリの内容をローカルリポジトリに反映します。

```
$ git fetch origin
```

git merge 【2つのブランチを統合】

2 つのブランチを統合します。

書式 現在使用中のブランチにマージ

```
$ git merge ブランチ名
```

本書で紹介 P.168, 194, 196

指定したブランチを、現在使用中のブランチにマージします。

```
$ git merge update-venue
```

書式 マージを中止し実行前の状態へ戻す

```
$ git merge --abort
```

応用

コンフリクトが発生して自動ではマージを完了できない状態になったときに、実行中のマージを中止し、実行前の状態に戻します。本書で説明したように、すぐコンフリクトを解消してマージできるとよいのですが、思わぬコンフリクトが発生してしまい 1 度元の状態に戻してから考えたいときや、すぐにコンフリクトを解消するのは難しくやっぱり中止したい事情があるときなどに使います。

```
$ git merge --abort
```

git以外のコマンド

pwd 【カレントディレクトリの絶対パスを確認】

カレントディレクトリの絶対パスを確認します。

書式

```
$ pwd
```

本書で紹介 P.47, 48

カレントディレクトリの絶対パスを確認します。

```
$ pwd
```

mkdir 【ディレクトリを作成】

ディレクトリを作成します。

書式 指定したディレクトリを作成

```
$ mkdir ディレクトリパス
```

本書で紹介 P.48, 49

指定したディレクトリを作成します。

```
$ mkdir ichiyasa
```

書式 ディレクトリを再帰的に作成

```
$ mkdir -p ディレクトリパス
```

応用

ディレクトリを再帰的に作成します。指定したディレクトリパスにディレクトリがない場合、存在しないディレクトリも含めて作成します。たとえば、ichiyasa ディレクトリ、study ディレクトリ、git ディレクトリがない場合、次のように実行すると、3 つのディレクトリを一気に作成します。

```
$ mkdir -p ichiyasa/study/git
```

ls 【ファイルやディレクトリを表示】

あるディレクトリに格納されているファイルやディレクトリを表示します。指定したディレクトリに格納されているファイルやディレクトリを表示します。ディレクトリを指定しない場合はカレントディレクトリの内容を表示します。

書式 カレントディレクトリ内のファイルとディレクトリを一覧で表示

$ ls

本書で紹介 P.49

カレントディレクトリに格納されているファイルとディレクトリを一覧で表示します。

```
$ ls
```

書式 指定したディレクトリ内のファイルとディレクトリを一覧で表示

$ ls ディレクトリパス

本書で紹介 P.50

指定したディレクトリに格納されているファイルとディレクトリを一覧で表示します。

```
$ ls ichiyasa
```

指定したディレクトリに格納されているファイルとディレクトリを一覧で表示します。

書式 すべてのファイルやディレクトリを一覧で表示

$ ls -a ディレクトリパス

本書で紹介 P.50

ls コマンドだけでは見えないすべてのファイルやディレクトリなどが表示されます。たとえば、Windows の Thumbs.db、macOS の .DS_Store や .gitignore ファイルは ls コマンドのみの実行では表示されませんが、-a オプションを付けると表示されるようになります。

```
$ ls -a ichiyasa
```

cd 【指定したディレクトリへ移動】

指定したディレクトリへ移動します。

書式

$ cd ディレクトリパス

本書で紹介 P.47, 51

指定したディレクトリへ移動します。

```
$ cd ichiyasa
```

clip / pbcopy 【クリップボードにコピー】

指定した内容をクリップボードにコピーします。Windows では clip、macOS では pbcopy を使います。

書式

$ clip < ファイルパス

本書で紹介 P.120 Windows

ファイルの中身をすべてコピーします。

```
$ clip < .ssh/id_rsa.pub
```

書式

$ pbcopy < ファイルパス

本書で紹介 P.120 macOS

ファイルの中身をすべてコピーします。

```
$ pbcopy < .ssh/id_rsa.pub
```

code 【Visual Studio Codeで開く】

ファイルやディレクトリを Visual Studio Code で開きます。ただし、このコマンドを使うには設定が必要なので、本書の手順に従ってインストールを済ませる必要があります（P.53 以降参照）。

書式 ディレクトリをVisual Studio Codeで開く

```
$ code ディレクトリパス
```

本書で紹介 P.132, 133

指定したディレクトリを Visual Studio Code で開きます。

```
$ code ichiyasaGitSample
```

書式 ファイルをVisual Studio Codeで開く

```
$ code ファイルパス
```

応用

指定したファイルを Visual Studio Code で開きます。

```
$ code ichiyasaGitSample/index.html
```

start / open 【エクスプローラーまたはFinderで開く】

指定したディレクトリをエクスプローラー（Windows）または Finder（macOS）で開きます。

書式

```
$ start ディレクトリパス
```

本書で紹介 P.134 Windows

エクスプローラーでディレクトリを開きます。

```
$ start ichiyasaGitSample
```

書式

```
$ open ディレクトリパス
```

本書で紹介 P.134 macOS

Finder でディレクトリを開きます。

```
$ open ichiyasaGitSample
```