

Gitで文書をチーム管理する方法

きっと分かる！
プログラムソースコード管理だけじゃない
GITの活用法



自己紹介

Name	弓削田公司（ゆげたこうじ）
Job	株式会社 MYNT 代表取締役
Web	https://myntinc.com
	https://blog.myntinc.com
	https://www.facebook.com/yugeta.koji
	https://x.com/yugeta_koji



Gitを知ろう！

～業務効率化のための第一歩～

GIT 使ってます？

このセミナーは、GITは聞いたことあるけど、
使ったことない人に
ちょっとだけ**興味**を持ってもらう事を
目的にしています。

GIT とは？

変更履歴を管理

誰がいつ何を変えたか記録する仕組み

分散型バージョン管理システム

- みんなが自分のPCで履歴を持てる仕組み
- ネットがなくても作業＆履歴管理ができる仕組み

GITの歴史

知っておくと、誰かに話したくなる（かもしれない）

1. Linusが3日でGitを作った

Linuxの開発者の Linus Torvalds が、
2005年4月にわずか3日ほどで Git の初期バージョン を書き上げた。

2. GITが作られた経緯

Linuxカーネルの開発で、
「BitKeeper」のライセンス問題が起きたため。

3. 名前の由来は自虐的ジョーク

「git」はイギリスのスラングで
「嫌なやつ」「バカ」という意味もあり、
Linusが自分自身を指してそう名付けたとも言われている。

4. Linuxの発言

Linusは「私はクソ野郎（Bastard）で、
私の作るツールはクソ野郎向けのツール(gitのこと)だ」と
笑いながら言っていた。

つまり強力で便利だけど、最初は扱いが難しかった。

GITのヒント

エンジニア以外でもGITが使えるアイデア**

GITは、エンジニアだけの道具ではありません。
あらゆる職種で、“変化するもの”や“共同作業”にかかわるなら、
誰でもその恩恵を受けることができます。

1. エンジニア（使ってない人いないよね？）

- ソースコードの管理とバグ修正の履歴追跡
- 機能追加ごとのブランチ運用で安全な開発
- チーム開発でのコンフリクト（衝突）解決やレビュー対応

2. 企画職・マーケター

- キャンペーン施策案や提案資料のバージョン管理
- 修正履歴を活用して「なぜこのアイデアになったか？」を遡れる
- チームメンバーで資料を共同編集し、レビュー履歴を共有

3. デザイナー

- UIデザインやロゴ案など、デザインファイルの変更履歴管理
- バージョン違いの比較や、複数案の分岐管理に便利
- フィードバックを記録し、反映した修正の流れを可視化

4. ライター・編集者

- 原稿や記事の校正履歴を残せる
- 編集前後の比較が簡単で、過去の表現にすぐ戻れる
- 複数人での執筆プロジェクトでも、作業分担と統合がスムーズ

5. 営業・セールス

- 提案資料や価格表のバージョン管理（クライアント別など）
- 修正日・修正者の記録が残るので、チーム間での認識齟齬を防げる
- 過去にどんな資料を使ったかを履歴から確認可能

6. 人事・総務

- 社内規定やマニュアルの履歴管理
- 全社配布文書を安心して更新・共有できる仕組み
- 社内向けプロジェクトのドキュメントもGitで管理すれば一元化が可能

7. 教育・研修担当者

- 教材のバージョン管理・カリキュラムの履歴管理
- 複数人で作る研修資料のレビュー&編集履歴が残せる
- 研修用リポジトリを配布して、受講者とやりとりすることも可能

事例紹介

東京大学工学部ホームページ

担当者から相談を受けてページ改善をした内容の紹介。

改善前ページ

| 東京大学 大学院工学系研究科 技術部 |

技術部TOP 東京大学 工学系研究科 お問い合わせ サイトマップ

メニュー

技術部長挨拶
技術部組織
技術発表会
東大総合技術本部
リンク
旧ホームページ

Google検索
※学外向けページのみ検索可能です

技術部TOP

- 2024年2月7日 (学内向け記事)
[2024-2025年度技術部調整室員\(技術職員選出\)および議長選挙開票結果のお知らせ](#)
- 2024年2月7日 (学内向け記事)
[2024-2025年度技術部長推薦調整室員信任投票開票結果のお知らせ](#)
- 2024年2月5日 (学内向け記事)
[2023年度第4回技術部会議事録掲載](#)
- 2024年1月22日 (学内向け記事)
[2024-2025年度技術部会議長選挙の投票について](#)
- 2024年1月22日 (学内向け記事)
[2024-2025年度技術部調整室員\(技術職員選出\)選挙の投票について](#)
- 2024年1月17日 (学内向け記事)
[2024-2025年度工学系研究科技術部会議長選挙の公示](#)
- ▲ 2024年1月17日 (学内向け記事)

学内専用メニュー

«技術部
«委員会
«基盤部門
«講習会実行委員会
«専門技術グループ
«ポータルサイト
掲載依頼

技術部

問題点

1. ページ内の情報量が多い。
2. スマホで見てもPCと同じ表示。
3. フレームワークを使わず、全てのページがHTMLファイルで作成。
4. ホームページの更新作業をFTPで行っている。

改善後ページ

The screenshot shows the homepage of the Tokyo University Graduate School of Engineering Technical Department. The header includes the university's logo and name, along with links for TOP, お知らせ (News), 技術部 (Technical Department), 業務内容 (Business Content), 技術発表会 (Technical Seminar), リンク (Links), お問い合わせ (Contact), and 学内TOP (Intra-University TOP). Below the header are four images: a lab setup, a microscope, glass bottles, and a DNA helix. A blue arrow points down from the 'お知らせ' link to a news section. The news section features a blue square icon with a white '国' character, followed by the word 'ニュース' (News). It lists three items: 1. '学内' (Intra-University) dated '2025-06-19' with the link '2025年度第1回技術部調整室会議議事録を掲載しました。'; 2. '学内' (Intra-University) dated '2025-04-30' with the link '2025年度調整室員と担当を掲載しました。'; and 3. '学外' (External) dated '2025-04-28'.

<https://www.ttc.t.u-tokyo.ac.jp/ttc/public/>

改善ポイント

1. ページデザインの修正
2. フレームワークを構築して、ヘッダ、フッダ部分と、コンテンツ部分を切り分け。
3. FTPではなく、GITを使って、オートデプロイ方式を構築。**(←コレ！)**

実施内容

1. ホームページデザイン、コーディングの再構築。
2. ホームページ担当者（東大技術職員7名）に対して、GIT教育を実施。

期間：半年ぐらい

- ・環境構築：全員のPC端末にGITをインストール
- ・コマンド学習：GITの基本コマンドをリファレンスに沿って説明＆ワーク
- ・運用ルール：ブランチやプッシュの仕方などを作成

3. リニューアル公開から、運用指導を実施

期間：現在半年が経過

作業中の問題点や課題など

1. GITの学習を終えても、運用をしてみると、最初はミスも多い。
2. 最初はGithubで構築したが、セキュリティとコスト面から、サーバー内にリモートリポジトリを構築して、オートデプロイ処理を構築。
3. コンフリクトに対して、手慣れてもらうために、テストで何度もコンフリクトさせた。

導入効果 1

以前は、相互に同じファイルをアップした際に上書きしてデータを消してしまうトラブルが頻発していたが、GITを導入することで、ミスの巻き戻しも可能で、履歴も確認でき、誰がいつアップしたデータなのかを認識できるようになった。

導入効果 2

チェック作業がほぼなくなり、コンフリクトチェックだけすれば良くなつたので、大幅な時短に繋がつた。

導入効果 3

Markdown言語を合わせて学習することで、テキストで資料作成などが簡易にできるようになった。

Markdownは、色々なメモ帳アプリなどでも使えるため、覚えていると通常の作業効率もアップします。

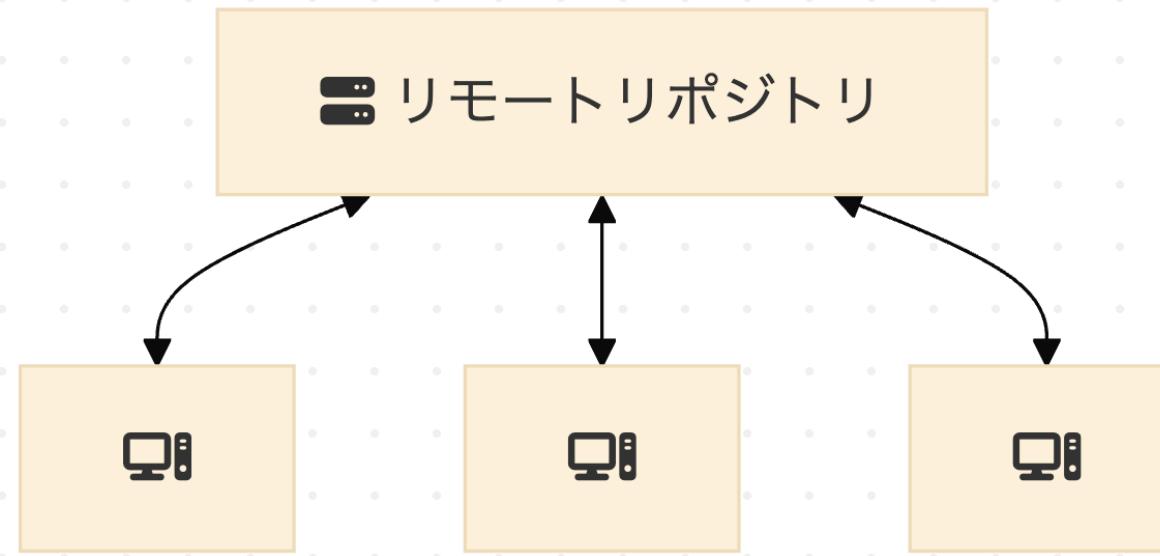
ちなみに、このセミナー資料もMarkdownのテキストで作成しています。

構成 : VScode + Marp (プラグイン) + Mermaid (図などの描画) + 画像ファイル

バージョン管理の仕組み

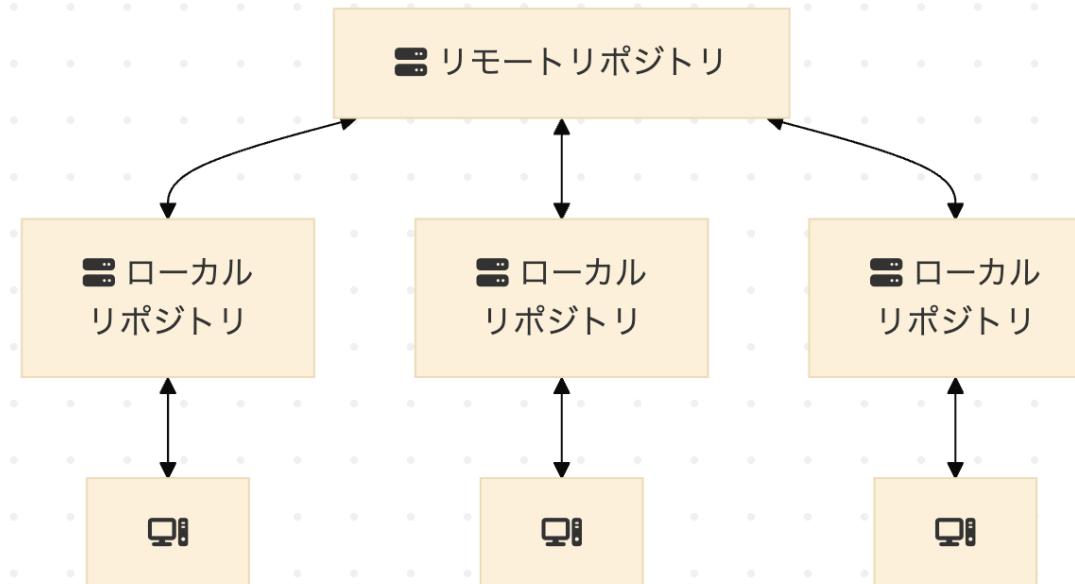
集中型と分散型の違い

バージョン管理 集中型管理の概念



svn , cvs , Perforce

バージョン管理 分散型管理の概念



Git

Gitに出てくる用語

リモートリポジトリ

共有リポジトリと言われる場合もある。
他の開発者と変更を共有するために使用されます。
多くの開発組織は、
Githubをリモートリポジトリとして使っている。

ローカルリポジトリ

各開発者は自分のパソコンに
リポジトリの完全なコピーを持ちます。
これにより、インターネット接続がない状態でも
開発作業を進めることができます。

ブランチ(branch)

ファイルの変更内容を、履歴として保存する操作。
「ここまで作業した」という区切りを記録することで、
後から見返したり、元に戻すことができるようになります。
コミットには「変更内容のメモ（メッセージ）」をつけます。

プッシュ (push)

ローカルリポジトリのコミット内容を
リモートリポジトリに送信する操作。
チームメンバーと変更を共有するために使います。

プル(pull)

リモートリポジトリから最新の変更を取得し、
ローカルリポジトリに取り込む操作。
他の人の作業内容を反映させたい時に使います。

マージ(merge)

別々のブランチで行われた変更を一つに統合する操作。
たとえば、機能追加用ブランチで作業した内容を
メインブランチにまとめるとときに使います。

コンフリクト (conflict)

複数のブランチで同じ部分が違う形で変更されたとき、
どちらを採用するか決められず、Gitが自動でマージできない状態。
手動で解決が必要になります。

クローン (clone)

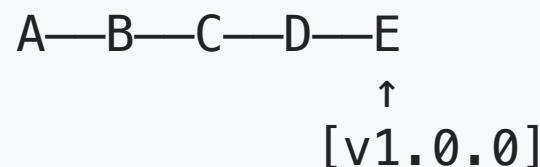
リモートリポジトリの内容を、
自分のパソコンにまるごとコピーして
ローカルリポジトリとして利用できるようにする操作。

タグ (tag)

特定のコミットに「ラベル（目印）」を付ける操作。

よく使われるのは、リリースバージョンの記録（例：v1.0.0）。

あとから「あの時点」をすぐに見つけるのに便利です。



ステージング

コミットする前に、「このファイルを保存対象にする」と選ぶエリアのこと。
変更をいったん“控え室”に置くようなイメージです。
その後、`git commit` で履歴に残します。

リベース (rebase)

あるブランチの変更履歴を、別のブランチの上に載せ替える操作。

ブランチの流れを直線的に整理したいときには使う。

ただし、使い方を間違えると履歴がぐちゃぐちゃになるため注意が必要です。

リベース前

```
main:      A—B—C  
feature:          \— D—E
```

リベース後

```
main:      A—B—C—D—E
```

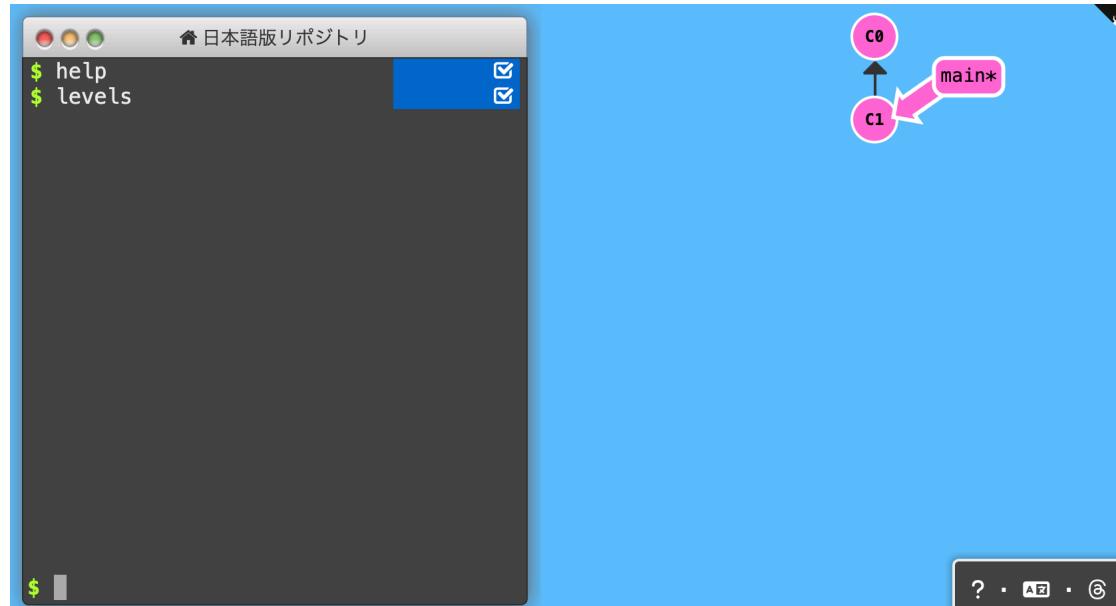
コミット

開発の分岐を管理するための機能。
新しい機能開発やバグ修正のために、
メインの開発ラインから分岐して作業を進め、
後でメインラインにマージすることができま

無料で学習できるサイト

#1 Learn Git Branching

<https://learngitbranching.js.org/>



#2 Git Immersion

<https://gitimmersion.com/>

Git Immersion

A guided tour that walks through the fundamentals of Git, inspired by the premise that to know a thing is to do it.

Git is a powerful, sophisticated system for distributed version control. Gaining an understanding of its features opens to developers a new and liberating approach to source code management. The surest path to mastering Git is to immerse oneself in its utilities and operations, to experience it first-hand.

#3 Oh My Git!

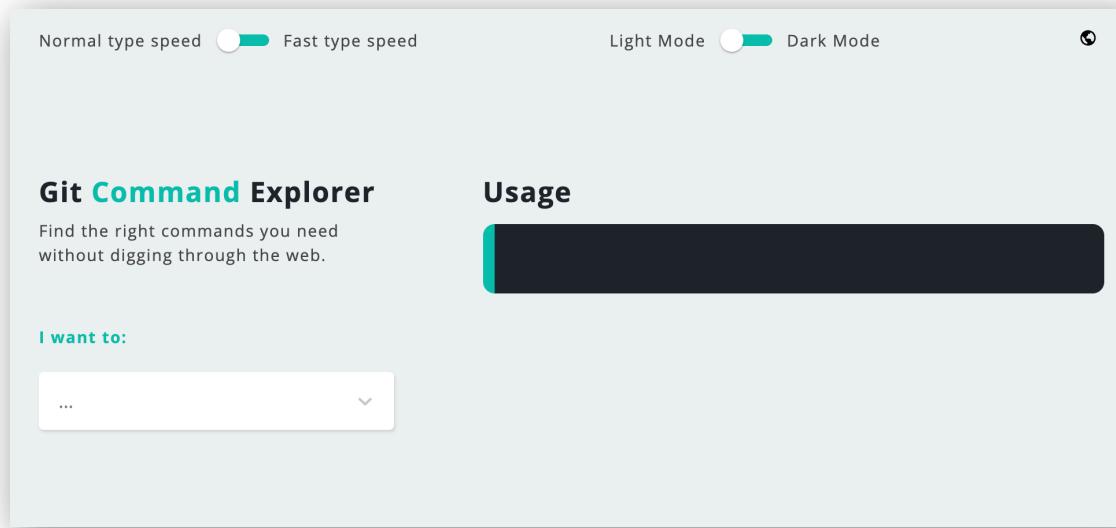
<https://ohmygit.org/>



GIT便利ツール

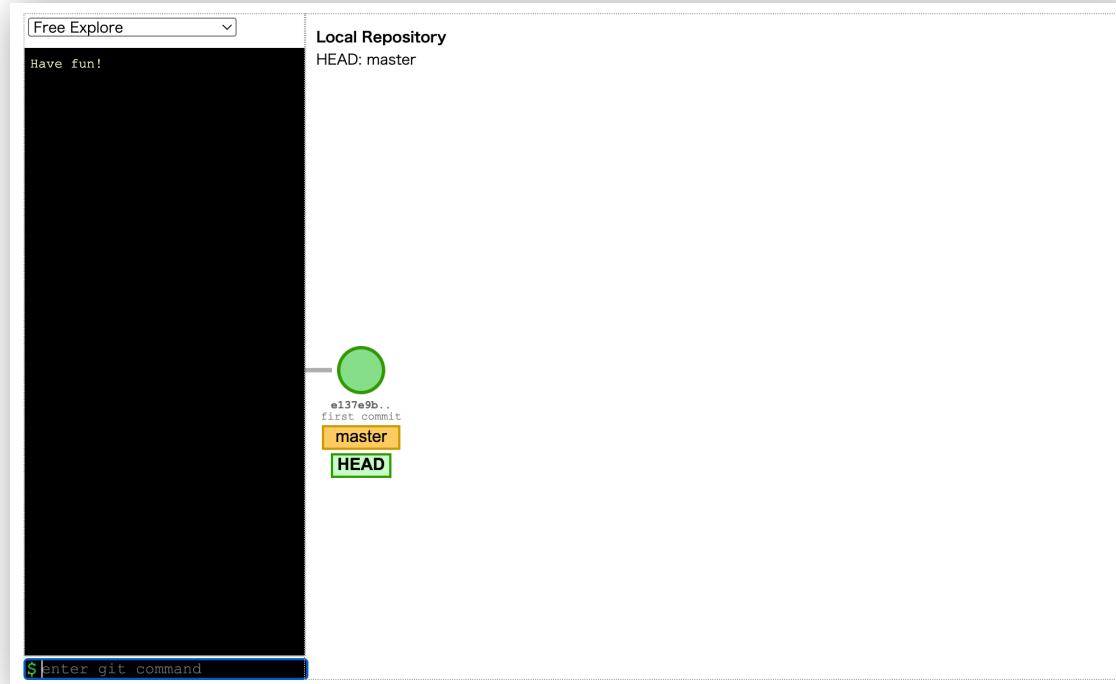
1. Git Explorer

<https://git.gaozih.com/>



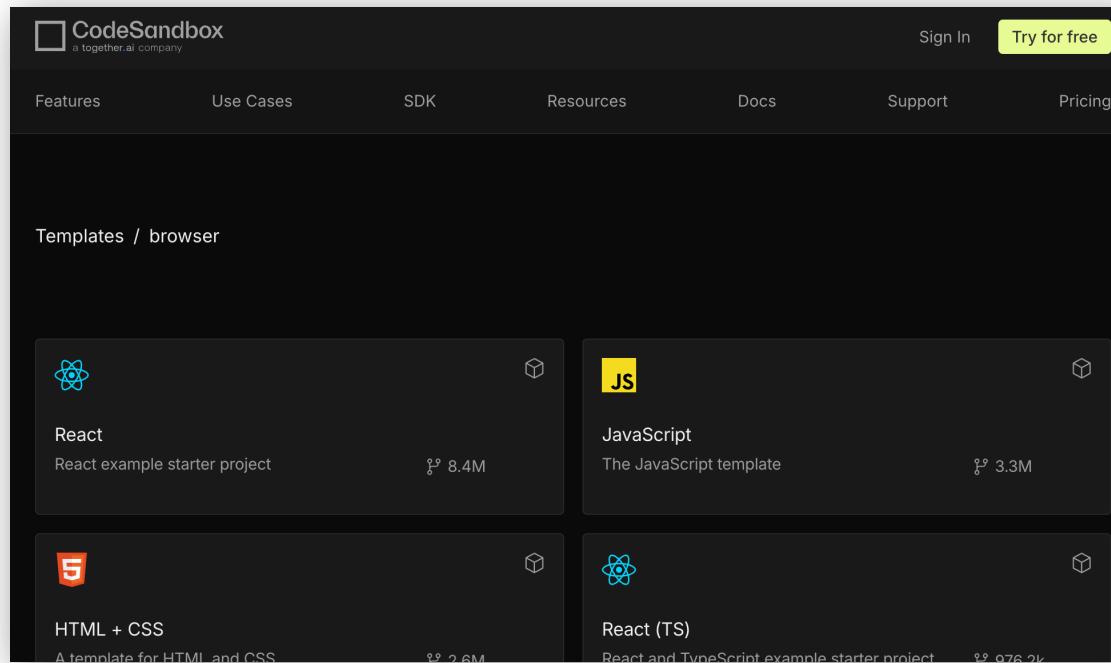
2. Git School – Visualizing Git

<https://git-school.github.io/visualizing-git/>



3. CodeSandbox

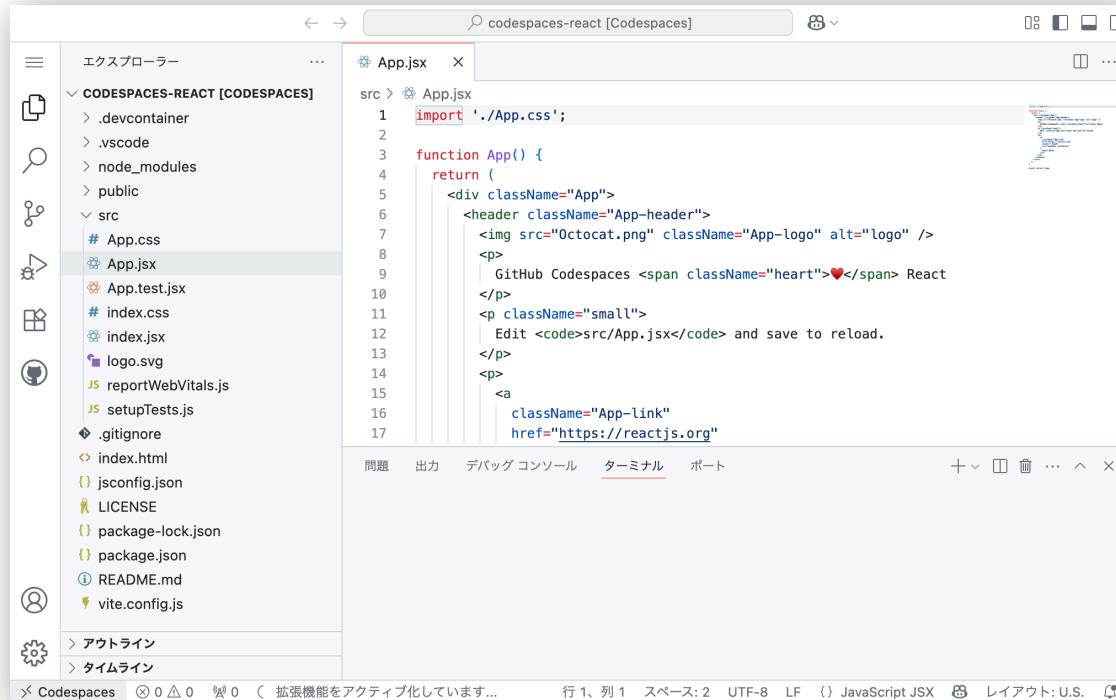
<https://codesandbox.io/>



ブラウザでコマンド操作

GitHub Codespaces

<https://github.com/features/codespaces>



The screenshot shows the GitHub Codespaces interface. On the left is a sidebar with icons for file operations like copy, search, and share. Below it is a tree view of the repository structure:

- エクスプローラー
- CODESPACES-REACT [CODESPACES]
 - .devcontainer
 - .vscode
 - node_modules
 - public
 - src
 - # App.css
 - App.jsx
 - App.test.jsx
 - # index.css
 - index.jsx
 - logo.svg
 - reportWebVitals.js
 - setupTests.js
 - .gitignore
 - index.html
 - jsconfig.json
 - LICENSE
 - package-lock.json
 - package.json
 - README.md
 - vite.config.js
- アウトライン
- タイムライン

The main area shows the content of the App.jsx file:

```
1 import './App.css';
2
3 function App() {
4   return (
5     <div className="App">
6       <header className="App-header">
7         
8         <p>
9           GitHub Codespaces <span className="heart">❤</span> React
10        </p>
11        <p>
12          | Edit <code>src/App.jsx</code> and save to reload.
13        </p>
14        <p>
15          <a
16            className="App-link"
17            href="https://reactjs.org"
18          >
```

At the bottom, there are tabs for 問題, 出力, デバッグ コンソール, ターミナル (which is underlined), and ポート.

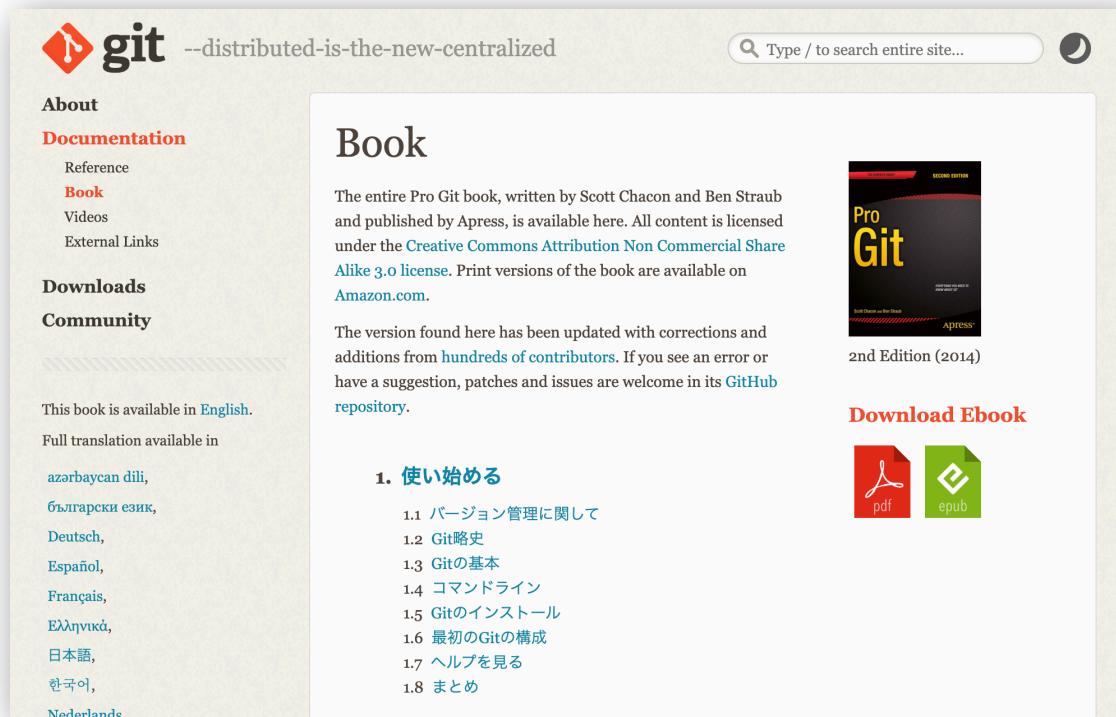
学習サイト

リファレンスサイト

Pro Git 日本語版

公式のGit解説書のオンライン日本語版

<https://git-scm.com/book/ja/v2>



The screenshot shows the "Book" section of the official Pro Git documentation. At the top, there's a navigation bar with the "git" logo, a search bar, and a refresh button. Below the navigation, there are links for "About", "Documentation", "Downloads", and "Community". The "Documentation" section is expanded, showing links for "Reference", "Book", "Videos", and "External Links". The "Book" link is highlighted in red. The main content area is titled "Book" and contains text about the availability of the entire Pro Git book in English, published by Apress under a Creative Commons Attribution Non Commercial Share Alike 3.0 license. It mentions that print versions are available on Amazon.com. Below this, there's a note about updates and corrections from contributors, and a GitHub repository link. On the right side, there's an image of the "Pro Git" book cover (2nd Edition, 2014) and a "Download Ebook" section with PDF and EPUB icons.

git --distributed-is-the-new-centralized

Type / to search entire site...

About

Documentation

Reference

Book

Videos

External Links

Downloads

Community

This book is available in [English](#).

Full translation available in

azərbaycan dili,

български език,

Deutsch,

Español,

Français,

Ελληνικά,

日本語,

한국어,

Nederlands.

Book

The entire Pro Git book, written by Scott Chacon and Ben Straub and published by Apress, is available here. All content is licensed under the [Creative Commons Attribution Non Commercial Share Alike 3.0 license](#). Print versions of the book are available on [Amazon.com](#).

The version found here has been updated with corrections and additions from [hundreds of contributors](#). If you see an error or have a suggestion, patches and issues are welcome in its [GitHub repository](#).

1. 使い始める

- 1.1 バージョン管理に関して
- 1.2 Git略史
- 1.3 Gitの基本
- 1.4 コマンドライン
- 1.5 Gitのインストール
- 1.6 最初のGitの構成
- 1.7 ヘルプを見る
- 1.8 まとめ

Download Ebook

 pdf  epub

導入編

自分のパソコンにインストールする方法

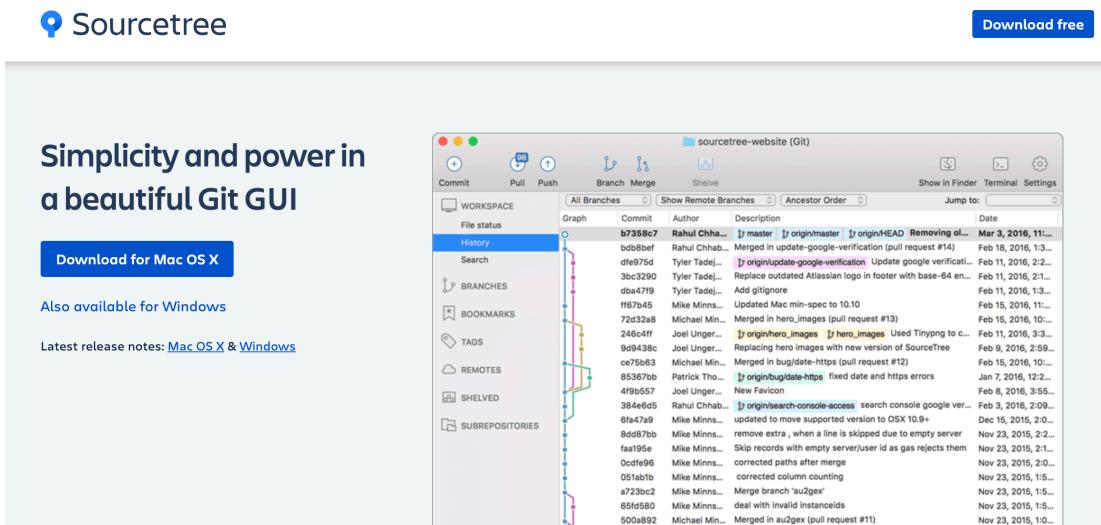
GUI? CUI?

- GUI(アプリケーション)は、見た目でわかりやすい。
- CUI(コマンド)は、全ての機能が使える。

GUIツール

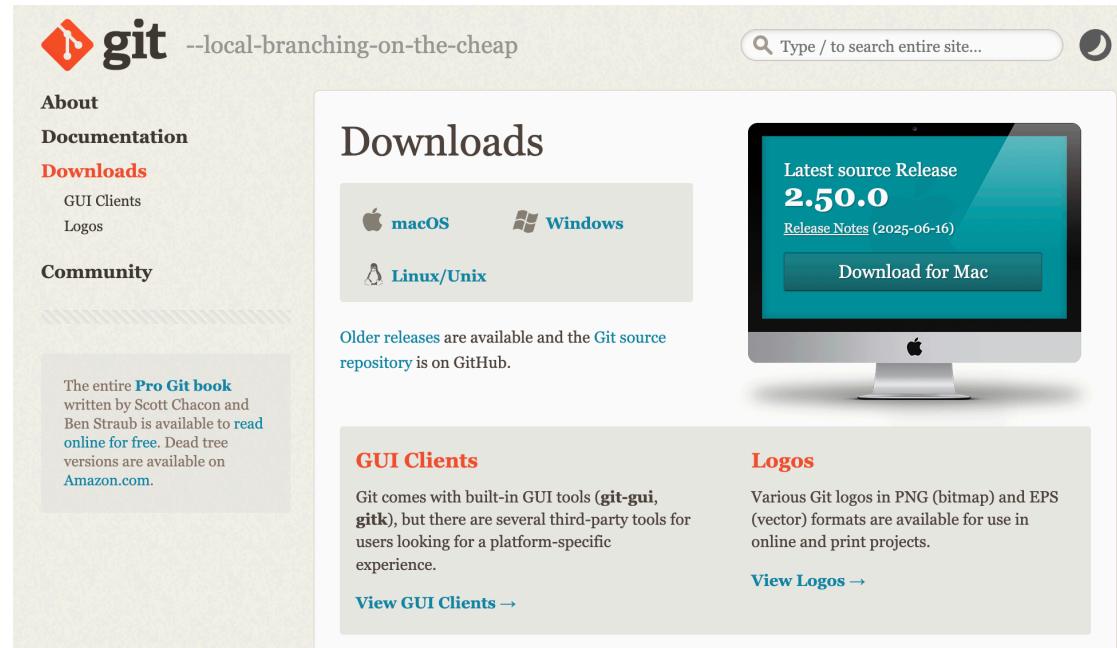
SourceTree

<https://www.sourcetreeapp.com/>



CUIツール

<https://git-scm.com/downloads>



基本操作

コマンド一覧

リポジトリ操作系

- git clone
- git init

ブランチ操作系

- git branch
- git checkout

ステージ操作系

- git staths
- git add
- git commit
- git tag

コミット操作系

- git push
- git pull
- git merge
- git fetch

履歴操作系

- git log
- git reset
- git revert
- git rebase

難易度高い系

- git cherry-pick
- git reflog

Githubのリポジトリ操作

簡単操作体験

使用ツール

このURLにアクセス

| <https://github.com/features/codespaces>

clone

githubから、リポジトリをローカルに clone (コピー) する

```
git clone https://github.com/yugeta/sample
```

ファイル操作をして、次のコマンド

新規ファイルを追加

ファイルの内容を更新

```
git add . ← ドットを忘れずに  
git commit -m '修正作業'
```

コミット内容を確認

```
git log
```

コマンドを実行して表示された文字を理解してみましょう。

活用事例

NasにGitリポジトリサーバーが搭載できる事例

NAS(Network Attached Server)に、
Gitサーバーがインストールできる機器があります。

- Synology
- Qnap (うちの会社はコレ)
- UGREEN

(他にも色々あります・・・)

NASにGitサーバーを構築するメリット

1. 会社（組織）内のみで管理できるので、情報漏洩の心配がない。
2. Githubは、容量制限や、各種セキュリティ制限があるが、
 そうした制限が一切ない。
3. Actionsなどの機能が無料で使い放題。
 (Actionsは、ホームページのオートデプロイなどで使う機能)

NASのGitサーバーデメリット

1. 社内や自宅に置いているため、電気代がかかる。
2. 設置作業や、保守管理などが自己責任。

オススメの方法

1. NASのバックアップにGithubなどのクラウドサービスを使う。
2. 運用ルールはクラウドサービスに合わせる。
3. Actionsは、NASサーバーで行うことで、無駄な金額発生を防ぐ。
4. 外部会社との連携はGithubで行う。
(クラウドでは、publicとprivateを使い分ける)

もちろん、Githubのみでの運用も問題ありません。

運用ルール例

GITをチーム運用する時に必要なルール

1. 個人開発向け：シンプル運用

- main ブランチのみで作業
- コミットメッセージは「何をしたか」が明確になるように記述
例：fix: ログイン時のバグ修正
- 毎日の作業終了後に git push
- タグでバージョン管理（例：v0.9）

2. 少人数チーム向け：軽量ブランチ運用

- main : リリース用ブランチ
- dev : 開発中の統合ブランチ
- 個人作業ごとに feature/ ブランチで作業
例 : feature/add-login
- 完了後、dev にプルリクエスト（レビュー）
- main へのマージは代表者のみ実施（週1など）

*プルリクエストは、Githubの機能です。

3. 中規模チーム向け：標準運用

(GitHub Flow風)

- main : 常にデプロイ可能な状態を維持
- 作業ごとに feature/ ブランチを作成
- プルリクエストベースでレビュー＆マージ
- GitHub Actions等で自動テスト連携
- main へのマージ後、自動デプロイ (CI/CD)

4. 大規模・多チーム向け：リリース＆長期保守運用

(Git Flow風)

- main : 本番用（タグ付きでリリース管理）
- develop : 次リリース向けの統合開発ブランチ
- feature/ : 新機能追加
- release/ : リリース準備用ブランチ
- hotfix/ : 本番での緊急修正
- 厳密なレビュー・CIテスト・承認フローあり

5. ドキュメント・非エンジニアチーム向け

- main : 本番用（タグ付きでリリース管理）
- develop : 次リリース向けの統合開発ブランチ
- feature/ : 新機能追加
- release/ : リリース準備用ブランチ
- hotfix/ : 本番での緊急修正
- 厳密なレビュー・CIテスト・承認フローあり

CIテストは、継続的自動テストの意味。

6. ドキュメント管理チーム向け

(例：広報・人事・マニュアル担当)

ブランチ構成

- main : 公開状態のマニュアルや文書を置く
- edit/○○ : 内容修正や更新ごとの作業ブランチ
例 : edit/2025-rule-update

ルール例

- 修正内容ごとに必ずブランチを分ける
- プルリクエストでチーム内レビューを行ってから main へマージ
- コミットメッセージは「何を変えたか」を簡潔に
例 : fix: 入社手続きの説明文を更新

7. 企画・営業資料チーム向け

(例：マーケター・セールス)

ブランチ構成

- main : 最新の正式提案資料など
- proposal/クライアント名 : 顧客別の資料バージョン
例 : proposal/ABC-corp

ルール例

- 資料提出前にレビューを行い、確認者がチェックしてからマージ
- バージョン番号タグで、提出資料を明確に記録
例 : v2025-07-05-ABC

推奨フォーマット

- .pptx もGitで管理可能
(バイナリなのでdiffは取れないが履歴保存には使える)



8. コンテンツ制作チーム向け

(例：ライター・編集者)

ブランチ構成

- main : 公開中のコンテンツ
- draft/タイトル : 記事や投稿の下書きブランチ
例 : draft/ai-tool-review

ルール例

- 下書きは1ブランチ1記事
- レビューを受けてから main にマージ、公開時にタグを付ける
例 : tag: blog-2025-07-10

運用補足

- Markdown形式で書くと、差分も確認しやすく便利
- 画像は /assets/ などにまとめて管理

9. 社内文書チーム向け

(例：総務・法務)

ブランチ構成

- main : 最新の全社向け文書・規定集
- review/○○ : 修正案や監査用の修正ブランチ
例 : review/契約書フォーマット改訂

ルール例

- 文章の修正は issue に起票してから対応
- 修正理由と背景をコミットメッセージに残す
例 : refactor: 法改正に合わせて条文を更新

セキュリティ

- クローズドなGitサーバー
(GitHub EnterpriseやGitLab、NASなど) を活用

10. 教育・研修資料チーム向け

プランチ構成

- main : 最新の公開済み教材
- update/年度やテーマ : 改訂用プランチ
例 : update/2025-spring-basic

ルール例

- 研修終了後のフィードバックを元に内容を更新
- バージョンタグ付きで教材配布履歴を管理
例 : v2.1-新人研修2025春

補足

- マークダウン教材とPDFをセットで管理することで、編集と配布の両立が可能

よくあるルール設定例（共通）

- コミットメッセージの書き方統一
例：[fix] バグ修正 / [add] ログイン機能追加
- 禁止事項：mainに直接pushしない、force pushしない
- レビュ一体制：最低1人以上のレビューでマージ可能
- 命名ルール：feature/xxx, bugfix/xxx, hotfix/xxx

コミットメッセージ例

目的

何を、なぜ、どのように変更したのかを明確にする
チームメンバー間の認識齟齬をなくす
あとで履歴を見ても「意味が通じる」ようにする

コミットメッセージ例

基本ルール

書き方の型（シンプル版）

```
php-template  
コピーする  
編集する  
<種別>: <変更内容の要約>
```

```
<b>例</b>  
fix: 誤字を修正  
add: 新しい提案書テンプレートを追加  
update: 2025年度の研修資料を最新版に更新
```

コミット種別 (prefix)

書き方の型 (シンプル版)

種別	意味・用途	例
add	新規追加 (資料・文書・機能)	add: 新しいFAQセクションを追加
fix	バグや誤字、内容ミスの修正	fix: 資料の日付表記ミスを修正
update	内容や表現の調整・更新	update: トーンを調整して説明を読みやすくした
remove	不要なファイル・記述の削除	remove: 古い仕様書を削除
refactor	構成・表現の整理 (意味は変えない)	refactor: セクション構成を整理
docs	ドキュメントの追加・修正	docs: READMEに注意事項を追記

NG例と改善例

NG例	改善例
修正	fix: 表記ゆれを修正
aaa、 test	add: 動作確認用テストファイル追加
更新しました	update: 年度表記を2025年に変更

コミットメッセージのルール化

ルールを守ると得られる効果

誰が見ても「何をしたか」がすぐに分かる
レビュー・調査がスムーズになる
ドキュメントとしても役立つログが残る

アクセス権限について

目的

- 間違った操作や意図しない変更からリポジトリを守る
- チームメンバーの役割に応じて、安全かつ効率的な運用を実現
- 「誰が何ができるか」を明確にし、トラブルを未然に防ぐ

権限ロール（役割）定義の例

ロール	できること	想定される対象者
管理者	すべて（設定変更、権限付与、削除、強制push）	プロジェクトマネージャー、技術責任者
メンテナー	ブランチ保護設定、マージ、タグ付け	チームリーダー、信頼された担当者
開発者	ブランチ作成、プルリク作成、編集	エンジニア、ライター、デザイナーなど
閲覧者	リポジトリの閲覧のみ	上長、外部レビューア、他部署関係者など

推奨ルール設定例（GitHub／GitLab等を想定）

ブランチ保護ルール（main, masterなど）

- 直接 push を禁止（強制）
- 誤操作による上書き防止
- プルリクエスト（Merge Request）経由のみマージ可能
- レビューを必須化
- レビュー2人以上で承認
- チームでの品質担保
- CIテスト通過必須（開発用の場合）

推奨ルール設定例（GitHub/GitLab等を想定）

書き込み権限の制限

main に push できるのは管理者・メンテナーのみ

develop, feature/* は開発者もpush可

docs/ フォルダのみ編集可能なチーム（非エンジニア用）を作成可能

非エンジニア向け配慮ポイント

Git初心者には限定的な権限を付与（例：閲覧・編集のみ）

docs用サブリポジトリを作って権限を分離

プルリクエスト時に説明テンプレートを用意

例：「何を、なぜ変更したか」「レビューしてほしいポイント」

役割変更や退職時の対応

リポジトリのオーナーが定期的にメンバーリストを確認
不要なアカウント・チームは速やかに削除または無効化
設定変更や削除操作のログ取得（GitHub Audit Log、GitLab Activityなど）

サンプル：アクセス権ルール表（小規模チーム向け）

操作	管理者	メンテナー	開発者	閲覧者
リポジトリ設定変更	✓	✗	✗	✗
main ブランチに push	✓	✓	✗	✗
プルリクエスト作成	✓	✓	✓	✗
プルリクのレビューとマージ	✓	✓	✗	✗
docs/ フォルダ編集	✓	✓	✓	✗
リポジトリの閲覧・クローン	✓	✓	✓	✓

補足：トラブルを防ぐヒント

運用ポリシーをREADMEやCONTRIBUTING.mdに明記する
Slackやメールと連携してPRの通知を飛ばす
週次でマージログやアクティビティをレビュー

教育研修の計画

目的

Gitの基本概念と操作を習得し、
業務ドキュメントや資料を安全かつ効率的に管理できるようにする。
チームでの共同作業時に起こる「上書きミス」や「履歴の混乱」を防止する。
非エンジニア職にも使いやすいGit運用習慣を根付かせる。

スケジュールサンプル（全4回）

回数	テーマ	内容概要
第1回	Gitの基本概念と操作体験	バージョン管理とは？、Gitの用語 clone・commit・push体験
第2回	実務で使えるGit操作	ブランチ作成 pull・merge コンフリクト対応の簡単な練習
第3回	チーム作業での運用ルール	プルリクエスト レビューの流れ コミットメッセージのルール
第4回	非エンジニア向け活用事例共有	ドキュメント管理、資料レビュー 共同編集の具体例 / 振り返りと質疑応答

使用教材・ツール案

オンライン演習：

GitHub Codespaces、GitPod、または仮想PC環境

スライド教材

Marp形式やGoogle Slidesなど

ハンズオン用の教材リポジトリ

例：sample-docs, training-branching

チートシート

Gitコマンド早見表

受講対象者とレベル分け

タイプ	対象職種	内容のレベル感
初心者	総務・人事・営業・マーケ	GUI中心操作、コミット・プルの体験
中級者	ライター・ディレクター・デザイナー	ブランチ運用・レビュー・コンフリクト対応
上級者	(補助) エンジニア、チームリーダー	補助スタッフ・質問対応係

学習目標（例）

Gitの役割やメリットを説明できる

自分の作業をコミット・プッシュして履歴を残せる

チームメンバーの変更をpullし、自分の作業と統合できる

ブランチやレビューを使った共同作業の流れを理解できる

研修を成功させるコツ

いきなりコマンド操作ではなく、「なぜ使うのか？」から始める
GUIツール（GitHub Desktop / GitKraken など）を併用
ミスしても安心な「演習用リポジトリ」を事前に用意
ハンズオンの後に「実際の業務でどう使えるか」を結びつける

オプション追加要素

マニュアル更新プロジェクトでGit活用の実演
コミットメッセージやPRレビューのワークショップ
自社ルール化ドキュメント (CONTRIBUTING.md) の作成ワーク

抵抗勢力への対応策ほか

よくある抵抗パターンとその背景

抵抗の声	背景・本音
「Gitって難しそう」「コマンド無理」	ITツールへの不安、過去の失敗体験
「Excelで十分」「今まで困っていない」	現状維持バイアス、変化への抵抗
「これ覚えるより本業が先」	優先度が見えにくく、業務負荷に感じている
「失敗したら怖い」	誤操作のリスクを過大視している
「Gitって技術者向けでしょ？」	自分に関係ないとと思っている

対応策と工夫

1. 「体験してみる」 → 理解が進む

演習を通して、「自分の作業に使える」と実感させる

例：研修資料の修正履歴がきれいに残っている例を見せる

対応策と工夫

2. GUIツールを活用する

GitHub Desktop / Sourcetree / Visual Studio Codeなど
「コマンド覚えなくていいんだよ」と伝えるだけで安心感が生まれる

対応策と工夫

3. 小さく始める

まずは一人で使う／ドキュメントだけで使うなど
成功体験を積んでもらい、他チームへ自然に波及

対応策と工夫

4. 上司・リーダーが先に使う

「現場がやれと言ってる」より「上の人気がもう使ってる」が効く
管理職が使えば、評価基準にも組み込みやすくなる

対応策と工夫

5. ミスしてもいい環境をつくる

研修用リポジトリは壊しても大丈夫な設計に
revert や reset を教えておけば「戻せる安心感」になる

対応策と工夫

6. 成果・変化を“見える化”する

Git導入前後でのドキュメント履歴の比較例を提示
「こういう場面で助かる」という具体例を共有

成功のカギは「共感」と「現場感」

「覚えることが多い」ではなく「やることが減る」と伝える
ITっぽさよりも「文書の整理術」「編集の見える化」として紹介
「あなたのためのツールです」と寄り添った説明が◎

おまけ：導入初期のプチ施策例

「コミットメッセージ大賞」などのゆるいイベント
Git用語かるた、Gitbingoなどの学習ゲーム
導入初月は“Git相談役”を決めておく
SlackやTeamsで「Gitの小ネタ」投稿を週1回

質疑応答

この際だから何でも聞いてみよう！

以上

ご静聴ありがとうございました。