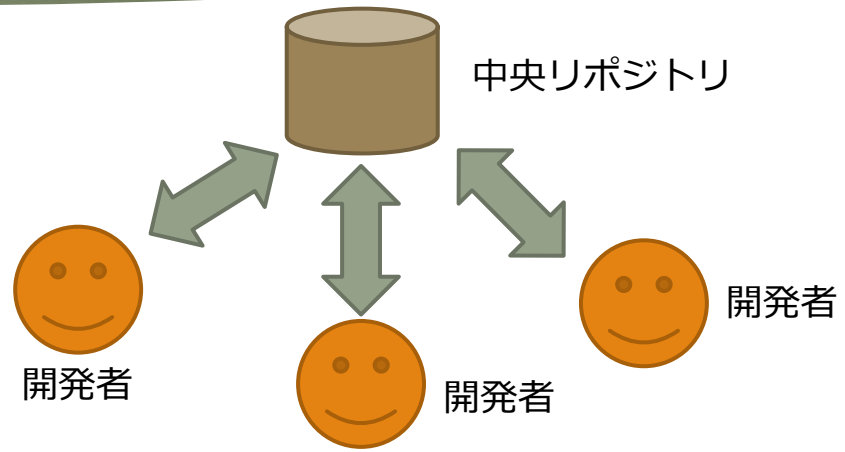
Subversionユーザーのための分散型SCM「Git」活用の勧め

2013年7月19日　初稿

基礎知識①：リポジトリの集中型と分散型

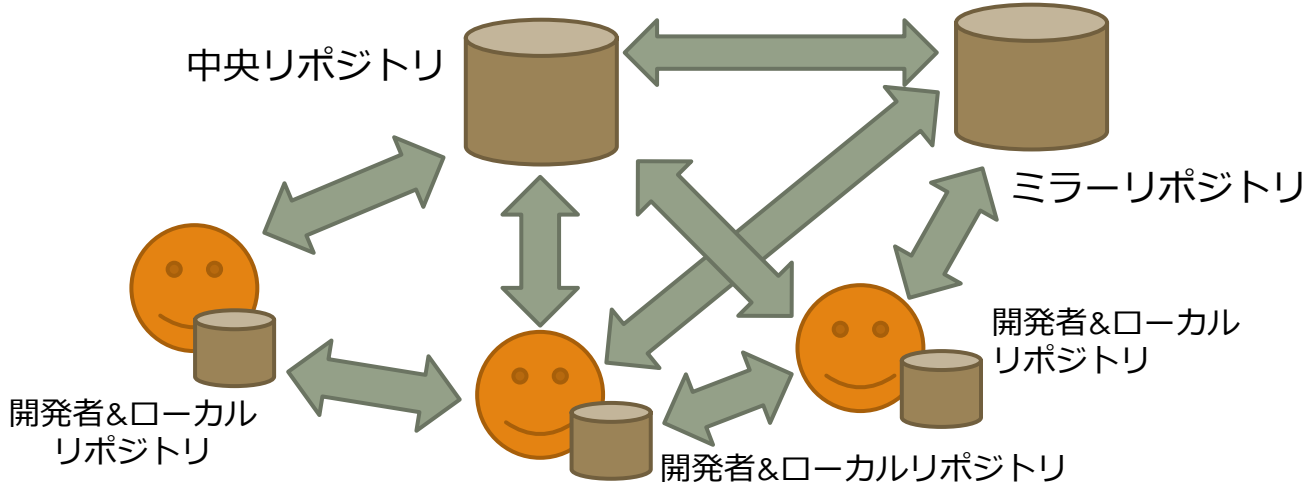
- **集中型（C/S型）**
  - 開発者チーム全体が一つの中央リポジトリを共通利用するタイプ。
  - 主な該当VCS：

    SVN(**S**ub**v**ersion), **CVS**(**C**oncurrent **V**ersions **S**ystem), **VSS**(Microsoft **V**isual **S**ource **S**afe), **TFS**(Microsoft Visual Studio **T**eam **F**oundation **S**erver), Perforce, Alien Brain など



- **分散型**
  - 各開発者がそれぞれローカルリポジトリを持ち、中央リポジトリやそのミラーリポジトリなどの多数のリポジトリを同期を取りながら利用するタイプ。
  - 主な該当VCS：

    Git(ギット), **Mercurial**(マーキュリアル), **Bazaar**(バザー), BitKeeper など

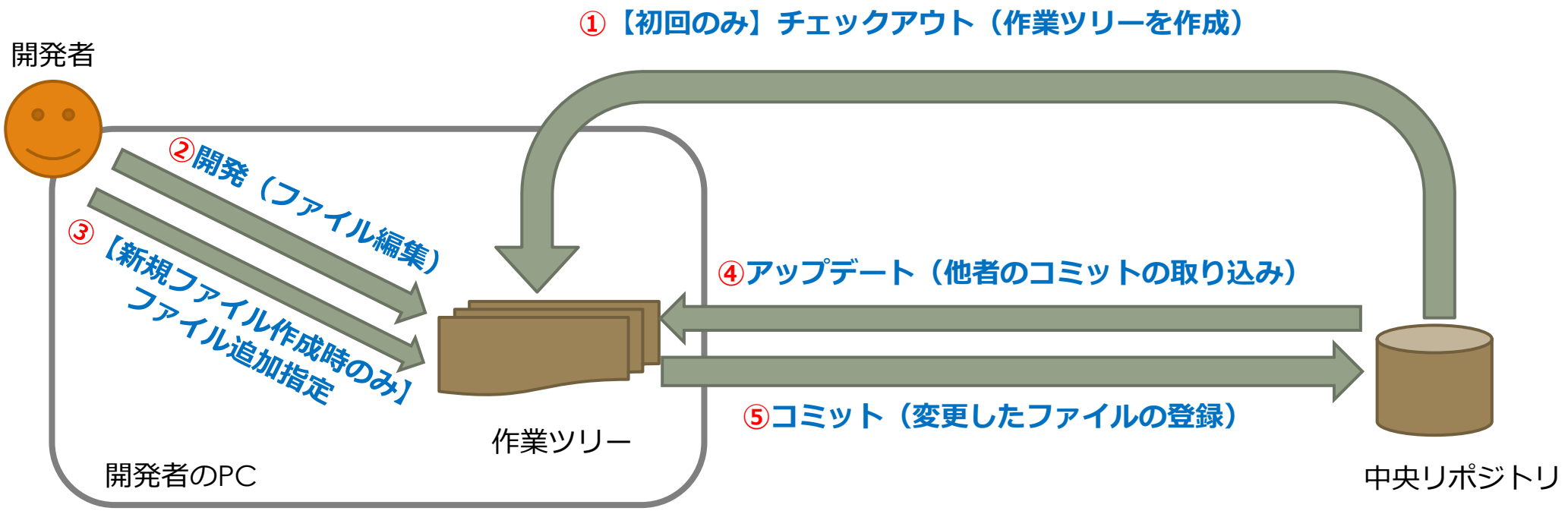


基礎知識②：VCSとSCM

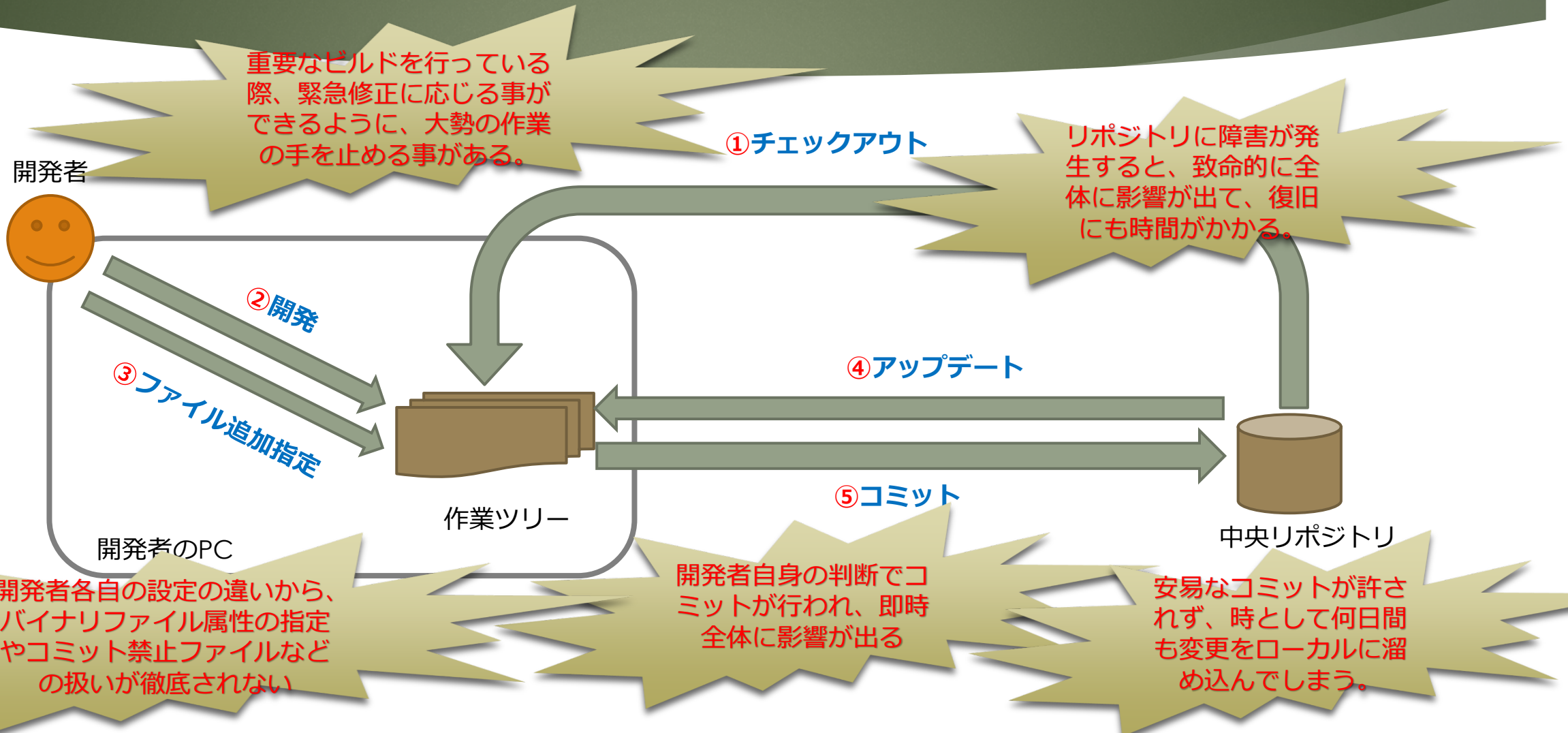
- **VCS**
  - **V**ersion **C**ontrol **S**ystem = バージョン管理システム
    - GitやSVNなどの、バージョン管理を行うシステムそのものを指す。

- **SCM**
  - **S**oftware **C**onfiguration **M**anagement = ソフトウェア構成管理
    - VCSのみならず、開発プロセスとその為の環境を含んだものを意味する。
    - タスク管理、BTS（Bug Tracking System = バグ追跡システム）、CI（Continuous Integration = 継続的インテグレーション = 自動ビルド、自動テストなど）、プルリクエスト／マージリクエスト、コードレビューといった、VCS以外の機能を含めている。
    - VCSと連携する主な製品としては、Track、Redmine、Jenkins（Hudson）などが、特に有名な物として挙げられる。
    - GitHubのように、VCSと一体となったタスク（Issue）管理、プルリクエストなどの機能を備えたホスティングサービスもある。

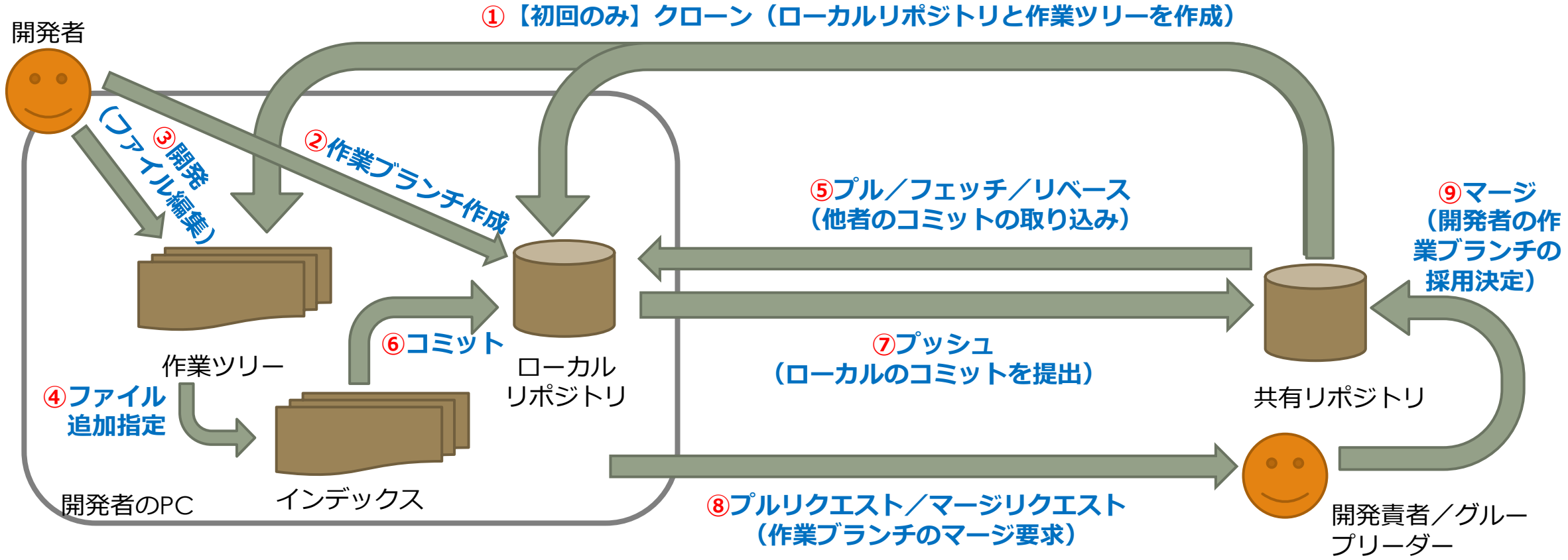
Subversionのワークフロー



Subversionの問題点



Gitのワークフロー


①【初回のみ】クローン（ローカルリポジトリと作業ツリーを作成）
開発者
②作業ブランチ作成
③開発（ファイル編集）
作業ツリー
④ファイル追加指定
インデックス
⑥コミット
ローカルリポジトリ
開発者のPC
⑤プル／フェッチ／リベース（他者のコミットの取り込み）
⑦プッシュ（ローカルのコミットを提出）
共有リポジトリ
⑨マージ（開発者の作業ブランチの採用決定）
⑧プルリクエスト／マージリクエスト（作業ブランチのマージ要求）
開発責者／グループリーダー

Gitの利点①

- **頻繁なコミットが可能**
  - 不安定なソースコードをコミットしても他者に影響が出ないので、ビルドが通らないような状態でも、小まめにコミットできる。
  - 開発ブランチを用いる事により、共有リポジトリにプッシュしても安全。
- **管理者を通したワークフローの健全化**
  - 開発ブランチのマージを管理者に委ねる事により、開発者自身以外の判断で正規のシステムへの組み込みを判断する事や、事前のコードレビューなどを行えるようにする。
- **開発者の作業の手を止めない／問題発生時はすぐに対処**
  - 作業ブランチにより、本流に影響を与えないワークフローの為、重要なビルドを行っている最中も個人の作業を継続できる。
  - 問題が発生したら、その時点の作業ブランチを一旦コミットし、本流に切り替えて、問題点の対処を行える。

Gitの利点②

- **コミットのキャンセル・再コミット**
  - Subversionなどと異なり、一度コミットした内容を修正して再コミットできる。
- **オフライン開発**
  - ローカルリポジトリへのコミットは、共有リポジトリがなくても可能な為、オフライン状態でも作業を継続できる。
- **遠隔地開発**
  - 遠隔地の拠点に専用の共有リポジトリ（ミラーリポジトリ）を設置して、拠点内のチームで完結する作業体制を取る事ができる。

Gitの利点③

- **リポジトリのバックアップ**
  - バックアップの為の特別な仕組みを用意しなくても、リポジトリのコピーがあちこちのローカルにあるので、普及しやすい。
- **コミット除外ファイル、ファイル属性をリポジトリに記録して共有**
  - リポジトリの先頭ディレクトリに特定のファイル名でそれらを定義できる為、ユーザーごとにファイルの扱いが異なるといった問題を防ぐ事ができる。

Gitの問題点

- 操作が複雑で覚えるべき事も多く、敷居が高い
- Subversionのように、自動で採番される一貫したシンプルなリビジョン番号がない
- RCS, CVS, Subversionのようなキーワード展開が出来ない
- ファイルロックで編集の競合を防ぐ仕組みがない
- 空のフォルダを扱えない
- ユーザーのアクセス権制御が難しい

Gitを活用すべきケース

- プログラム開発（ソースコード管理）全般
- 遠隔地（拠点間）での共同開発を行う場合
- ソーシャル開発を行う場合

Gitを活用すべきでないケース

- Excelなどのバイナリファイルを複数の開発者が変更しなければならない場合
- 既にSubversion等を使用していて、前述のような問題が出ていない現場の場合
- 多くのスタッフに対して、なるべく簡潔化した操作マニュアルを提示できない場合
- 問題発生時に十分スタッフのサポートができる体制が取れない場合

以上

SUBVERSIONユーザーの為の
分散型SCM「GIT」活用の勧め