事前質問

本編でふれられなかったもの

Gitはどのような場面で有効か？

- チーム開発
  - 複数名での開発
  - 組織をまたいだ開発
- 継続的な開発
  - 個人開発であっても
- マシンをまたいでツールを使う
  - dotfilesの管理とか
- デファクトスタンダード
  - みんな使ってる
  - Gitに関連した便利なものが多い

Gitとは何なのか？何に使えるのか？

- バージョン管理ツール
- チーム開発で使いやすい
- 論文執筆とかにも使われる
  - pushするたびにLaTeXのビルドをする，スペルチェック
  - Pull Requestでレビュー
  - 共同執筆
- GitHubやGitLabのようなGitのホスティングサービスではタスク管理も出来る
  - ソースコードに紐づくのが便利
- Git操作にフックして何かを自動で処理したり
  - slackに通知を流したり

なぜ「ジット」ではなく「ギット」とよぶのか？

- Gitはもともとはネットスラング
- Gitを作ったリーナスの発音は「ギット」のほうが近いように聞こえる
    - 検索すると講演動画とかが出てきます

ポートフォリオとしてのGit

- GitHubやGitLabで公開しているリポジトリがあれば教えて下さいということ
  - 自分がDeepFlowに入るときはgist（GitHubの機能）で履歴書を送りつけた
- 採用目線でいうとものすごく情報量が多い
- エンジニアとしてはアピールしやすい
  - 自分が何ができるのかをコードを示した上でアピールできる
  - 口で「できます」という以上の説得力がある
- 人に読まれると思ってコーディングする経験
- GitHubでいろんなソースコードを読んで参考するといい

Git講習会に興味はないが，DeepFlowには興味がある

- 興味をもってもらえて嬉しいです，ありがとうございます
- Gitという技術自体はおもしろい要素は多いと思うんで，ソースコードを部分的に読んだりするのはおもしろいと思います
- DeepFlowは物理シミュレーション（数値計算）がメイン事業です
  - 物理・数学・アルゴリズムなどあらゆるものを駆使して課題解決する
  - 頭を使うヘビーなディスカッションが多い
  - 雰囲気は会社というよりかは研究室に近い
  - それなりにみんなの専門分野（得意領域）がずれている
  - ソースコードの設計をどうするのか，開発効率をどうやってあげるのかという議論も多い
  - 新しいものもすぐに取り入れて試してみる文化（とりあえず手を動かす）
  - 博士の人も活躍してる

ブランチ運用・コミット粒度

- チーム開発での話は本編でしたので割愛
- 個人開発ではどうしているか？
- masterブランチとフィーチャーブランチで運用
  - 個人開発であってもブランチは切る
  - masterブランチでは必ずテストに通ることを保証する
  - 時間たってから使おうと思ったときに強い
- コミットの粒度
  - ビルドにとおったとき，テストしたいとき，頭を切り替えるとき
  - バグったときに適切に戻れることを意識してまずは細かくコミットしていく
- マージ時のコミットの整理
  - ある程度はまとめる
  - どのコミットでもテストに通るように
  - Conventional Commitsで適切なタグをつけることができるかというのが一つの目安

同一ユーザーの複数マシン間での同期はできるか？

- 自動で同期はできないが，手動ではできる
- リモートブランチ(GitHub) を経由する
  - 研究室のPCでリモートにpush
  - 自宅PCでリモートからcheckoutやpullする
- 他のメンバーと共有するときも同じようにする
  - A「なんとか機能のコードの共有お願いします」
  - B「このブランチにpushしました」
  - A「ありがとうございます」
  - masterやdevelopブランチを経由せずにやりとりはできる
- バックアップ目的としても頻繁にpushする癖をつけておく

プログラム開発のノウハウ，大規模開発の事例

- 大きいテーマなのでキーワードだけです。検索してみてください
  - SOLID原則
  - クリーンアーキテクチャ
  - なんとか駆動開発（いろいろある TDDとかDDDとか）
  - アジャイル開発
- サービス運用に向くコードと研究に向くコードは性質が違うので注意
  - 安定依存の原則（変化の激しいものには依存しない）
  - 研究過程での試行錯誤の激しいコードには依存できない
- elkurage開発（10万行くらい）
  - 開発状況にあわせて依存関係を変更するリファクタリングを行ってきた
  - 依存関係を整理しているので，物理のことよくわからなくても開発に参加できてる