Refactored Git Rebase Script: New Workflow and Functionality

Overview

The `git_rebase.sh` script has been refactored to provide a more structured, phased, and interactive workflow. The original script combined all operations (cloning, comparison, and rebase/merge) into a single, sequential loop. The refactored version separates these concerns into distinct functions, allowing for better readability, maintainability, and most importantly, user interaction at critical junctures. The new workflow follows three main phases:

1. **Cloning Repositories**: Ensures all necessary repositories are present locally.
2. **Comparing and Listing Changes**: Identifies and displays repositories with differences between the source and target branches.
3. **User Confirmation and Conditional Rebase/Merge**: Prompts the user for confirmation before proceeding with any Git operations that modify branches (creating feature branches, merging, and pushing).

Detailed Breakdown of Refactoring

The script is now organized into several functions, each responsible for a specific part of the overall process.

1. `clone_repos()` Function

This function is responsible for the initial setup, ensuring all specified repositories are cloned locally. It iterates through the `REPO_URLS` array and checks if each repository's `.git` directory exists. If not, it proceeds to clone the repository. This phase is designed to be idempotent; it will not re-clone repositories that are already present.

Key Features:

- **Idempotent Cloning**: Only clones repositories that are not already present locally.
- **Clear Output**: Provides messages indicating whether a repository is being cloned or if it's already present.
- **Error Handling**: Reports if a cloning operation fails but continues processing other repositories.

Code Snippet:

Bash

```
clone_repos() {
  echo ""
  echo "📦 Starting repository cloning process"
  echo
"================================================================================"
  for REPO_URL in "${REPO_URLS[@]}"; do
    REPO_NAME=$(basename "$REPO_URL" .git)
    CLONE_DIR="$WORK_DIR/$REPO_NAME"

    if [ ! -d "$CLONE_DIR/.git" ]; then
      echo "  Cloning $REPO_NAME..."
      if ! git clone --quiet "$REPO_URL" "$CLONE_DIR"; then
        echo "  ❌ Failed to clone $REPO_URL"
      else
        echo "  ✅ Cloned $REPO_NAME"
      fi
    else
      echo "  ✅ $REPO_NAME already cloned"
    fi
  done
  echo
"================================================================================"
  echo "✅ Repository cloning process completed."
}
```

2. `compare_and_list_changes()` Function

This function is dedicated to comparing the `SOURCE_BRANCH` and `TARGET_BRANCH` for each repository and identifying those with actual changes. It also incorporates the logic to ignore `Jenkinsfile` changes for specific repositories, as requested previously. This function does *not* perform any modifications to the branches; its sole purpose is to report the status.

Key Features:

- **Branch Comparison**: Performs `git diff` between `origin/$TARGET_BRANCH` and `origin/$SOURCE_BRANCH`.
- **Jenkinsfile Exclusion**: For specified repositories (`product-search-api-impl` and `rate-schedule-api-impl`), it filters out `Jenkinsfile` changes. If only `Jenkinsfile` changes are present, the repository is considered to have

no changes.

- **Clean State**: Ensures a clean working directory (`git reset --hard`, `git clean -fd`) and fetches latest updates (`git fetch --all`) before comparison.
- **Status Reporting**: Prints a formatted table showing the existence of source/target branches and whether changes were found.
- **Returns Changed Repositories**: Collects the names of repositories with detected changes and returns them as a space-separated string.
- **Early Exit**: If no changes are found in any repository, the script exits after this phase.

Code Snippet:

Bash

```
compare_and_list_changes() {
  echo ""
  echo "🔁 Starting branch comparison and listing changes"
  echo
"================================================================================"
  echo "| Repository                                    | Src   | Tgt   |
Changes |"
  echo "--------------------------------------------------------------------------
----"

  CHANGED_REPOS=()

  for REPO_URL in "${REPO_URLS[@]}"; do
    REPO_NAME=$(basename "$REPO_URL" .git)
    CLONE_DIR="$WORK_DIR/$REPO_NAME"

    if [ ! -d "$CLONE_DIR/.git" ]; then
```

```
      continue
    fi

    cd "$CLONE_DIR" || { echo "❌ Failed to enter '$CLONE_DIR'"; continue; }

    git reset --hard &>/dev/null
    git clean -fd &>/dev/null
    git fetch --all &>/dev/null

    git rev-parse --verify "origin/$SOURCE_BRANCH" &>/dev/null
    SRC_EXISTS=$?
    git rev-parse --verify "origin/$TARGET_BRANCH" &>/dev/null
    TGT_EXISTS=$?

    SRC_COL="No"
    TGT_COL="No"
    CHG_COL="N/A"

    if [ $SRC_EXISTS -eq 0 ]; then SRC_COL="Yes"; fi
    if [ $TGT_EXISTS -eq 0 ]; then TGT_COL="Yes"; fi

    if [ "$SRC_COL" == "Yes" ] && [ "$TGT_COL" == "Yes" ]; then
      DIFF=$(git diff origin/$TARGET_BRANCH..origin/$SOURCE_BRANCH)

      JENKINSFILE_IGNORE_REPOS=(
        "product-search-api-impl"
        "rate-schedule-api-impl"
      )

      IGNORE_JENKINSFILE=false
      for IGNORE_REPO in "${JENKINSFILE_IGNORE_REPOS[@]}"; do
        if [ "$REPO_NAME" == "$IGNORE_REPO" ]; then
          IGNORE_JENKINSFILE=true
          break
        fi
      done

      if [ "$IGNORE_JENKINSFILE" == "true" ]; then
        DIFF_NO_JENKINSFILE=$(git diff
origin/$TARGET_BRANCH..origin/$SOURCE_BRANCH -- ":!Jenkinsfile")
        if [ -z "$DIFF_NO_JENKINSFILE" ]; then
          DIFF=""
        fi
      fi

      if [ -n "$DIFF" ]; then
        CHG_COL="Yes"
        CHANGED_REPOS+=("$REPO_NAME")
```

```
      else
        CHG_COL="No"
      fi
    fi

    REPO_COL=$(printf "%-45s" "$REPO_NAME")
    SRC_COL=$(printf "%-6s" "$SRC_COL")
    TGT_COL=$(printf "%-6s" "$TGT_COL")
    CHG_COL=$(printf "%-8s" "$CHG_COL")

    echo "| $REPO_COL | $SRC_COL | $TGT_COL | $CHG_COL |"

    cd "$WORK_DIR"
  done

  echo
"================================================================================"
  if [ ${#CHANGED_REPOS[@]} -eq 0 ]; then
    echo "✅ No repositories have changes."
    exit 0
  fi
  echo "✅ Found changes in the following repositories:"
  for REPO in "${CHANGED_REPOS[@]}"; do
    echo "  - $REPO"
  done
  echo ""

  echo "${CHANGED_REPOS[@]}" # Return the list of changed repos
}
```

3. `perform_rebase_and_merge()` Function and User Confirmation

This is the final and most critical phase. After the `compare_and_list_changes` function identifies repositories with changes, the script will prompt the user for confirmation. Only if the user explicitly types `yes` (case-insensitive) will the `perform_rebase_and_merge` function be called for the identified repositories.

Key Features:

- **User Interaction**: Prompts the user for confirmation before making any changes to branches.
- **Conditional Execution**: The rebase/merge process only proceeds if the user confirms.

- **Targeted Operations**: The function receives the list of changed repositories as arguments, ensuring that operations are only performed on the relevant repositories.
- **Branch Management**: Handles checking out or creating the `FEATURE_BRANCH`, merging the `SOURCE_BRANCH`, and pushing the `FEATURE_BRANCH`.
- **Conflict Detection**: Reports merge conflicts and skips the repository, requiring manual intervention.
- **Status Reporting**: Provides clear output on the action taken for each processed repository.

Code Snippet:

Bash

```
perform_rebase_and_merge() {
  local repos_to_process=("$@")

  if [ ${#repos_to_process[@]} -eq 0 ]; then
    echo "No repositories to process for rebase/merge."
    return
  fi

  echo ""
  echo "🚀 Starting rebase and merge process for selected repositories"
  echo
"================================================================================"
  echo "| Repository                                    | Action
|"
  echo "------------------------------------------------------------------------
----"

  for REPO_NAME in "${repos_to_process[@]}"; do
    CLONE_DIR="$WORK_DIR/$REPO_NAME"

    echo "📂 Navigating into directory '$CLONE_DIR'..."
    cd "$CLONE_DIR" || { echo "❌ Failed to enter '$CLONE_DIR'"; continue; }

    # Clean up any existing changes
    git reset --hard &>/dev/null
    git clean -fd &>/dev/null

    # Fetch all branches
    echo "🔄 Fetching latest updates..."
```

```
    git fetch --all &>/dev/null

    ACTION="Skipped"

    echo "🌿 Checking out feature branch '$FEATURE_BRANCH'..."
    if git show-ref --quiet "refs/heads/$FEATURE_BRANCH"; then
      git checkout "$FEATURE_BRANCH" &>/dev/null
    elif git show-ref --quiet "refs/remotes/origin/$FEATURE_BRANCH"; then
      git checkout -b "$FEATURE_BRANCH" "origin/$FEATURE_BRANCH" &>/dev/null
    else
      echo "🚫 Feature branch '$FEATURE_BRANCH' does not exist locally or
remotely. Creating from '$TARGET_BRANCH'..."
      git checkout -b "$FEATURE_BRANCH" "origin/$TARGET_BRANCH" &>/dev/null
    fi

    echo "🔀 Merging '$SOURCE_BRANCH' into '$FEATURE_BRANCH'..."
    git fetch origin "$SOURCE_BRANCH" &>/dev/null
    git merge origin/"$SOURCE_BRANCH" --no-edit &>/dev/null

    if [ $? -ne 0 ]; then
      echo "⚠️ Conflict detected during merge in '$REPO_NAME'. Manual
resolution required."
      ACTION="Conflict"
      cd "$WORK_DIR"
      continue
    else
      echo "✅ Merge completed successfully."

      # Push the feature branch
      if git push origin "$FEATURE_BRANCH" &>/dev/null; then
        ACTION="Merged & Pushed"
      else
        ACTION="Merged but Push Failed"
      fi
    fi

    REPO_COL=$(printf "%-45s" "$REPO_NAME")
    ACTION_COL=$(printf "%-20s" "$ACTION")

    echo "| $REPO_COL | $ACTION_COL |"

    # Return to base directory
    cd "$WORK_DIR"
  done
  echo
"================================================================================"
  echo "✅ All selected repositories processed."
}
```

```
# Ask user for confirmation
if [ -n "$CHANGED_REPOS_LIST" ]; then
  echo "Do you want to proceed with creating feature branches and rebasing
the code for the above listed repositories? (yes/no)"
  read -r USER_CONFIRMATION

  if [[ "$USER_CONFIRMATION" =~ ^[Yy][Ee][Ss]$ ]]; then
    perform_rebase_and_merge ${CHANGED_REPOS_LIST[@]}
  else
    echo "Operation cancelled by user."
  fi
else
  echo "No changes detected, no rebase/merge needed."
fi
```

New Workflow Summary

1. **Initialization**: The script starts by defining variables and then immediately calls `clone_repos` to ensure all repositories are available locally.
2. **Change Detection**: Next, `compare_and_list_changes` is called. This function performs all necessary Git operations to identify which repositories have changes between the `SOURCE_BRANCH` and `TARGET_BRANCH`, respecting the `Jenkinsfile` exclusion rule. It then prints a summary table and lists the changed repositories.
3. **User Decision**: If any changes are detected, the script pauses and prompts the user: "Do you want to proceed with creating feature branches and rebasing the code for the above listed repositories? (yes/no)"
4. **Conditional Execution**:
   - If the user types `yes`, the `perform_rebase_and_merge` function is invoked, passing the list of changed repositories. This function then proceeds with checking out feature branches, merging, and pushing for each of those repositories.
   - If the user types anything else (e.g., `no`, `n`, or just presses Enter), the script prints "Operation cancelled by user." and exits without making any further modifications.

5. **Completion**: The script provides a final message indicating the completion of the process.

This refactored approach provides a clearer separation of concerns, enhances user control, and makes the script more robust and easier to understand.