GitHub

github.com

About

- Version control system (VCS)
- Owned by Microsoft
- Terminal commands
- Android Studio
- SourceTree

Commit style guide

- Origin name
    - Options: *root, trunk, main,* etc.
    - Avoid *master*
        - Change the default branch for existing projects
            - *See: Setting the default branch*
            - Remove *master* from Android Studio: *Preferences* > *Version Control* > *Git* > Add or remove branch name
- Use the imperative tense
- Subjects
    - Initialize: Start of a new repository
    - Feature: A new user feature
    - Utility: New architecture pattern, configurations, etc; No user feature change
    - Refactor: Updating/improving existing code
    - Design: Updating layouts, colors, shapes, and sizes; Same overall functionality
    - Fix: A bug resolution
    - Documentation: Changes to documentation
    - Test: Adding tests, refactoring test; No production code change
    - Release: Include version number
- Body
    - Details
    - Bullet points using '-' or '*'
- Footer: Issue tracker ID
- *Resources*
    - Udacity Nanodegree Style Guide
    - How to Write a Git Commit Message - *Chris Beams*
    - How to Write Good Commit Messages: A Practical Git Guide, *FreeCodeCamp*
    - How Square writes commit messages, *Square*

Setup

Repositories (Repos)

Usernames

- Changing your GitHub username
- Changes that are automatically updated
    - References to repositories will automatically redirect.
    - The redirects will stop working if a new owner of the previous username creates repos with the same name.
- Changes that cannot be updated
    - Links to the user profile
    - @mentions using the old username
    - Links to gists that include the old username
    - Profile README: Managing your profile README

Creating new repos

1. Create a repo on GitHub.
2. Enable version control system (VCS) in the code editor.
3. Set local origin.

Working with existing repos

- *See*: *Adding an existing project to GitHub using the command line*, *Documentation*

1. Clone projects with *ssh*, not *HTTPS* project*, or* Download from GitHub.
2. Set local origin.

Edit the repo description

- Repo landing page > Gear button in the top-right > Description

Email

- Setting your commit email address, *locally*
- Adding an email address to your GitHub account
    1. Use private email in commits
        a. *Settings* > *Emails* > Select *Keep my email addresses private*
        b. Add generated email by Setting your commit email address.
        c. Past commits
            1. How to amend several commits in Git to change author
                a. Use *-f* in command
                b. Set *--global* and non-global
            2. *git push -f*
    2. Troubleshoot: Your push would publish a private email address error
- Why are my contributions not showing up on my profile?

SSH key and token

About

- Setup required once per device.
- Connecting to GitHub with SSH

Steps

1. Checking for existing SSH keys
    a. If existing, follow Adding a new SSH key to your GitHub account
    b. If new, follow Generating a new SSH key and adding it to the ssh-agent
        i. Then, Adding a new SSH key to your GitHub account
2. Restart terminal, restart the computer if a terminal restart doesn't work.

Troubleshooting SSH key

- Failed authentication
    1. Make sure project is SSH'd connected to GitHub (Android Studio and SourceTree work with SSH token)
        a. Connect to project: *git remote set-url origin git@github.com:AdamSHurwitz/retrorecycler.git*
    2. Remove existing SSH key: *cd .ssh*
        a. *rm id_rsa*
        b. id_rsa.pub
    3. Add new SSH key to ssh-agent
- Repeated passphrase re-entry, *StackOverflow*

One-time token

1. Generate one-time token to use from machine
    - *GitHub > Settings > Developer Settings > Personal access tokens* (save the token to use in Android Studio and SourceTree)
2. Add Token access to Android Studio
    - *Preferences > Version Control > GitHub > Auth Type: Token*
3. If using corporate account use corp domain for host
    - i.e. *github.corp.companyName.com*

Contribute to other projects

Steps

1. Select *Fork*
2. Download/clone locally.
3. Configure the fork to sync with the original repository: Fork a repo > Configuring Git to sync your fork with the upstream repository (Step 7: Use *git remote add upstream*)
4. Checkout branch/create a new branch.
5. Add commits and push to origin.

6. Submit a pull-request (PR) to upstream.
    a. Go to the forked repository.
    b. Select *New pull request*.
    c. Make changes to the PR with commits and pushes to the submitted branch
7. Continually keep the repository fork up-to-date with changes in the original repository: Syncing a fork

Resources

- Commit history
    - Commits in a forked branch do not show in the history until a PR is made.
        - *See: Why are my contributions not showing up on my profile? > Commit was made in a fork*
    - Duplicate repo to show commit history
        - *See Duplicating a repository*
            - Use *HTTPS*
        - The new repo can be decoupled from the upstream branch

Gitflow branching

Resources

- Gitflow Workflow, *Atlassian*
- A successful Git branching model
- danielkummer.github.io/git-flow-cheatsheet

About: Gitflow → manages new features, fixes, releases, and support branches.

Implementation

1. Install Homebrew then GitFlow on computer.
2. Pull updates from QA and Start a new feature by starting a new Branch from QA.
3. Build feature.
4. Pull from QA (if changed).
    a. Select *feat* branch.
    b. Right-click QA > merge *QA* into *feat* branch.
    c. Push local *feat* branch to remote branch.
5. GitHub site > QA Branch > New Pull > **compare** *feat* branch to QA > Create Pull Request.

Branching

1. Create Branch.
2. Test new Branch with master.
    a. Rebase/Merge master into new Branch → right-click master Branch > rebase/merge master into new Branch.

i. Handle merge conflicts → Working > Determine if conflicts are changes you'd like to keep or discard.
ii. Test new feature.
iii. Move file(s) from working tree to staged > Click file to view end state of merge.
iv. Click Commit > Continue Rebase.
v. Force push to remote repo.

b. Merge new Branch into master.
c. Identify feature Releases / Tags → Push to remote.

Squash commits

[The GitHub Blog - Squash your commits]

Manual

1. Reset to one before the *first* commit, reset *on merge commit* (consecutive commits much easier to squash, don't try squashing merged commits)→ Soft Reset *keeps local changes*
2. Commit *locally.*
3. In the terminal cd into the project and force push → **git push -f**
4. Fetch to refresh the view of branch status.

Rebase

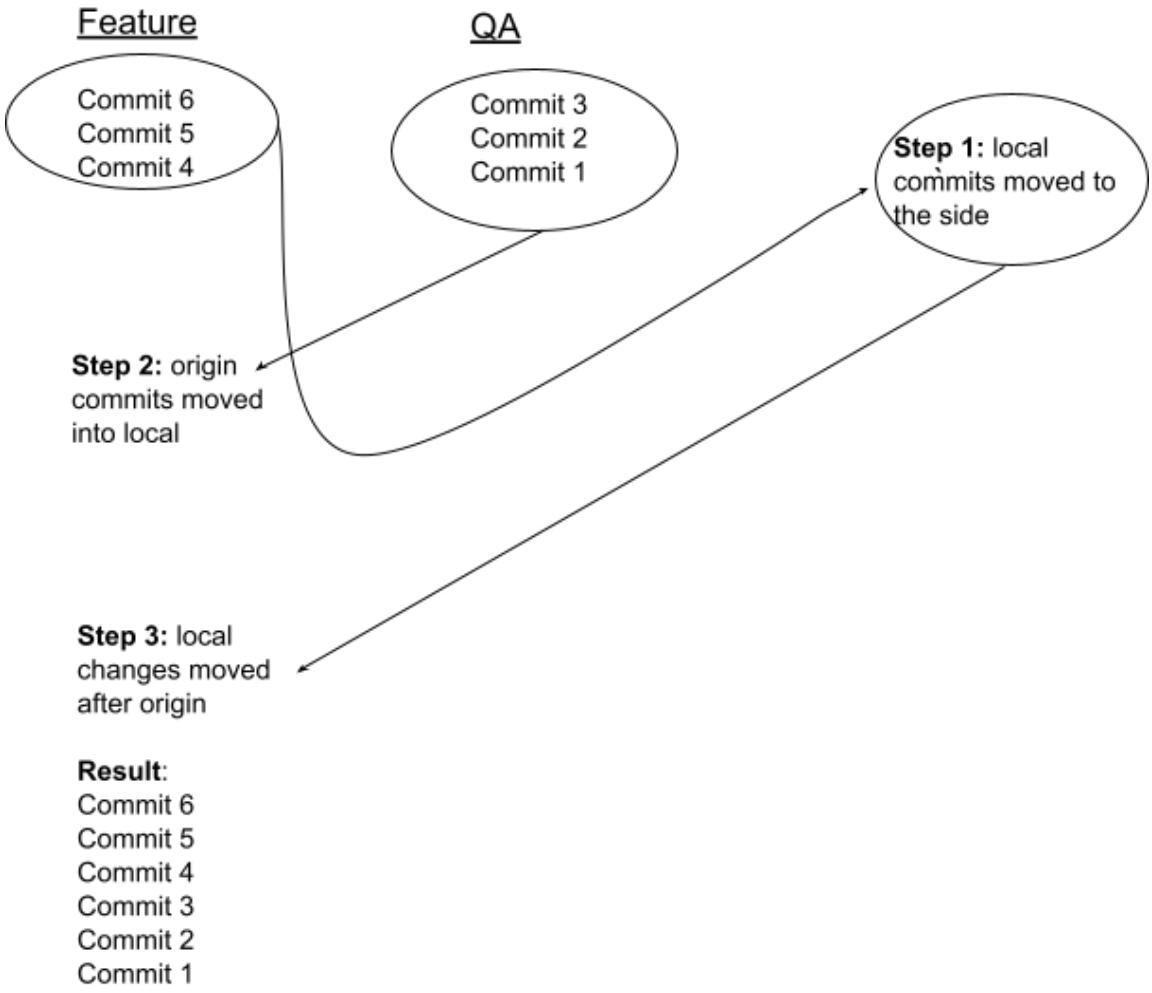
About

Rebase advantages

1. More readable.
2. One less commit since QA does fast forward, not merge.

Implementation

1. Squash local commits.
2. Rebase QA into feature (*Checkout in Feature branch*).



3. Handle merge conflicts.
4. Commit > Continue Rebase (*locally*).
5. Force push.
6. Merge (fast-forward) Local into Origin (*Checkout in QA*) → Must do manually by checking out QA and then merging feat branch into QA (*always Rebase feat branch before fast-forward*).

Debug

Uncommitted code: Use **Reset**, which goes to the last commit.

Committed code

1. Need to Edit.
    a. **Reset > *Soft*:** Re-opens commits to make changes/uncommit files > commit *locally* > force push.
    b. **Revert Hunks:** Creates new commit mirroring change > commit + push mirror.

2. Need to Undo: Use **Reverse commit…:** creates new commit mirroring the opposite > push.
3. Start from Scratch: Delete the branch.

Remove first/root commit

1. Modify an existing root commit: How do I git rebase the first commit?
2. Make the current Git branch a main branch

Fail-safe - Reflog

- *See: Atlassian Tutorials - Refs and the Reflog*
- About: Records almost every change you make in your repository, regardless of whether you committed a snapshot or not. You can think of it is a chronological history of everything you've done in your local repo.
- Implementation
    1. In terminal**:** *git reflog* to view > *git checkout HEAD@{numberOfHead}*
    2. Output: Most recent items at the top, i.e.:
        a. *400e4b7 HEAD@{0}: Checkout: moving from master to HEAD~2*
        b. *0e25143 HEAD@{1}: Commit (amend): Integrate some awesome feature into `master`*
        c. *00f5425 HEAD@{2}: Commit (merge): Merge branch ';feature';*
        d. *ad8621a HEAD@{3}: Commit: Finish the feature*
    3. Optional: Create a new branch once you're back to safe code: *git checkout -b <new-branch-name>***.**

'Detached Head': VCS > Git > Branches > Local > Origin/Master > Checkout

Reset Local Branch to Here > HardReset?

1. reload project: yes
2. remove root: yes
3. add to git: no

error: failed to push some refs to 'git@github.com/...'

- Attempt to push in the IDE interface.

Permission denied

- E.g. *git@github.com: Permission denied (publickey).*
- Need to fix the SSH key and/or token.

Related topics

Guides

- GitHub Git Cheat Sheet
- scotch.io cheat sheet

- easyGitGuide
- using git, *Squarespace*
- git book
- How to Use Git and GitHub, *Udacity*
- Git Auto-completion: git-scm.com - Git Basics Tips and Tricks
- Gists
    - Add multiple files: *Add file*.
    - Link to lines within multiple file gist: Select the line number(s) while holding *shift* and the url will update.

Merge conflicts

- About merge conflicts
- Solutions
    - Resolve simple conflicts in the GitHub web app: *Resolve conflicts* button
    - Resolve complex conflicts locally in a code editor.
    - Create a new branch
        - Pull the original repository's changes.
        - Make a new commit.

File templates

- Repository templates: Creating a repository from a template
- Open source license: Adding a license to a repository
- Ignore files
    1. Create *gitIgnore*
    2. Use GitHub template: *Create new file* > Name file *.gitIgnore* > Select *Android* template.

Pull request types

- *Create a merge commit*
    - Creates an additional commit.
- *Squash and merge*
    - Adds the branch's commits on top of the base branch
    - Does not create an additional merge commit
    - Works well if the branch's commits are squashed before the pull request.
- *Rebase and merge*
    - Works well if the branch's commits are squashed before the pull request.
    - It's best to resolve all merge conflicts and rebase manually to avoid breaking changes.

CherryPick

- Used to merge one commit into the current branch.

Chrome extension

- Octotree

Publish Markdown content

Open info

This is not technical advice. Always read the official documentation and do your own research.