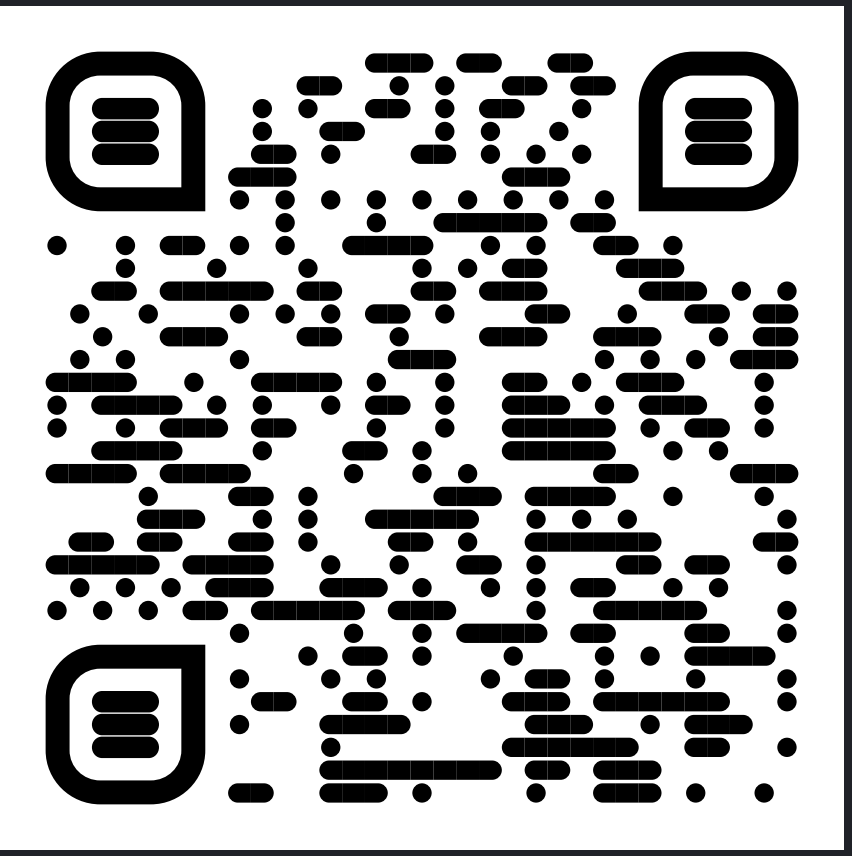
ESPRESSIF

Git & Workflow

Introduction of the Git Workflow in Espressif

IDF CORE TEAM



Overview 📖

- Get started with Git
- Professional in Git
- Submit Merge Request

Get Started with Git 🐾

> Git is a free and open source **distributed** version control system with *speed* and *efficiency*

GitHub VS GitLab

- Web-based Git repository
- GitLab: **Internal** develop, **CI/CD**
- GitHub: **Community** feedback, **Action**
- Issue tracking
- Code review

Linux

Mac

Windows


```
# Debian-based distribution
sudo apt install git

# Fedora
sudo yum install git

# Arch
sudo pacman -S git
```

```
# MacPorts
sudo port install git

# Homebrew
brew install git
```

Download installer for Windows

- `ls -la ~/.ssh` to see if you already have a public SSH key (e.g. *id_rsa.pub*)
- If not, create a new public and private key pair with command **ssh-keygen**
- Add your private key to *ssh-agent* with command **ssh-add ~/.ssh/id_rsa**
- Copy the contents of **id_rsa.pub** file to GitHub/GitLab/Bitbucket...

tldr

```
# get common usage on 'git diff'
tldr git-diff
```

man

```
# get all available information on 'git status'
git status --help # same as 'man git-status'
# get a quick help on 'git checkout'
git checkout -h
```

- Navigate to the directory you want to place under version control
- Create an empty Git repository: `git init`, this creates a hidden folder **.git**, which contains the entire history and configuration for the project

```
mkdir oh_my_project && cd oh_my_project
git init
```

```
cd /path/to/my/repo
git config user.name "Your name"
git config user.email "xxx@espressif.com"
```

- It will store the setting inside the individual repository: `/path/to/your/repo/.git/config`
- Advanced practice 💯: Fore Git to look for your identity only within a repository's settings

```
git config --global user.useConfigOnly true
```

- Check what files Git will add to your repository with command `git status`
- Review the resulting list of files, tell Git which of the files to place into version control (avoid adding files with confidential information)

```
git status
git add <file/directory name #1> <file/directory name #2> <...>
```

- If all files in the list should be added to version control

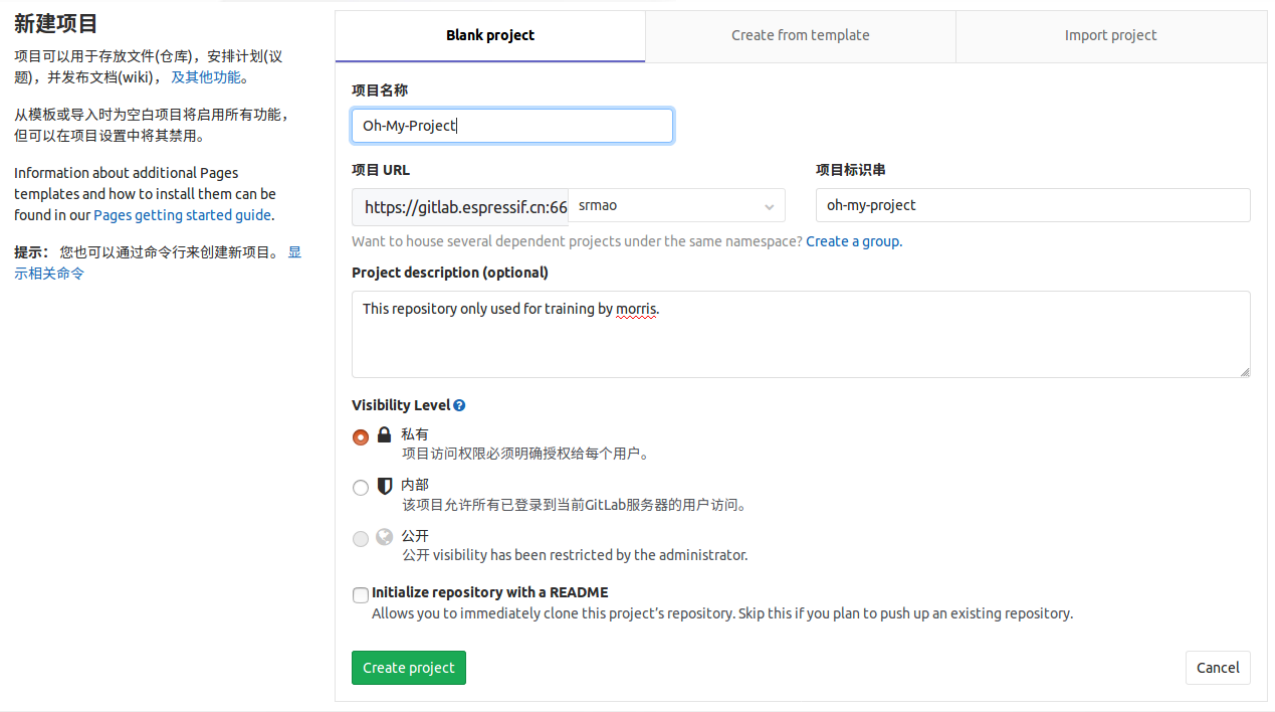
```
git add -A
```

- Commit all files that have been added, along with a commit message

```
git commit -m "add README.md"
# if you omit the -m parameter,
# your default editor will open,
# and you can edit and save the commit message there
```

- A commit is like a save or **snapshot** of your entire project
- You can now push it to a remote repository, and later you can jump back to it if necessary

Create a repository on remote



- To add a new remote, use the `git remote add` command in your local repository
- The `git remote add` command takes two arguments:
  - A remote name, e.g. **origin**
  - A remote URL

```
git remote add origin ssh://git@gitlab.espressif.cn:27227/srmao/oh-my-project.git
```

- If you want to change the URL of a remote

```
git remote set-url origin ssh://git@bitbucket.server.com:7999/username/project.git
```

- Show a list of existing remotes: `git remote -v`

- Copy your local repository to the remote
- Adding `--set-upstream` (or `-u`) created an upstream reference which is used by argument-less Git commands (e.g. `git pull`)

```
git push --set-upstream origin master
```

Professional in Git 

- List last 10 commits logs, on a single line: `git log --oneline -10`
- Log for a range of lines within a file: `git log -L 1,5:README.md`
- To see the log in a prettier graph-like structure

```
git log --decorate --oneline --graph -20
```

- Colorize logs

```
git log --pretty=format:'%C(red)%h%Creset -%C(yellow)%d%Creset %s %C(green)(%cr)%C(yellow)<%an>%Creset'
```

  - `%C(color_name)` option colors the output that comes after it
  - `%h` abbreviates commit hash
  - `%Creset` resets color to default terminal color
  - `%d` ref names
  - `%s` subject [commit message]
  - `%cr` committer date, relative to current date
  - `%an` author name

- Search for **changes** in **lines containing** specific string

```
git log -G "#define MODEM_COMMAND_TIMEOUT_MODE_CHANGE"
```

- Search **commit string** in git log

```
git log --all --grep "esp_modem"
```

- Filter logs

```
git log --since '3 days ago'
git log --author=morris
```

- Show the commits that are on `foo` branch but not on `master` with command:

```
git log master..foo
```

- Avoid typing pretty big commands by creating aliases:
  - with the command line

```
git config --global alias.ci "commit"
git config --global alias.st "status"
```

  - with the `~/.gitconfig` file

```
[alias]
    ci = commit
    st = status
    gl  = log -n 20 --date-order --format=\"%Cgreen%h %Cred[%ci] %Creset
    <%an>%C(yellow)%d%Creset %Creset %Cgreen%s %Creset \"
```

- If you have changed a lot of files in the directory, rather than listing each one of them, you could use:
  - `git add --all` to add all changes
  - `git add .` to add all changes, *not including files that have been deleted*, from the top-level directory and subdirectory
  - `git add -u` only add files which are currently tracked ("updated")

- Unstage a file that contains changes

```
git restore --stage <file>
```

- Show staged changes

```
git diff --cached
```

- Stage deleted files

```
git rm filename
```

- To delete the file from git without removing it from disk

```
git rm --cached filename
```

- For files that you want never under version control, create a file named `.gitignore` **before** staging
  - Can created in top level directory or sub-directories
  - Rules will apply recursively to all files and sub-directories

```
# ignore files called '.config'
.config
# ingore directories
bin/
# ignore files by extension
*.o
# matches all files 'foo.txt' in 'bar' and all sub-directories
bar/**/foo.txt
# ignore all .a files but lib.a
*.a
!lib.a
```

- Create an empty folder
  - Create the required directory and add a `.gitkeep` file to the folder
- Clean up ignored files

```
git clean -Xn # display a list of ignored files
git clean -Xf # remove the previously displayed files
```

- Find files ignored by `.gitignore`

```
git status --ignored
```

- Ignore subsequent changes to a file without removing it

```
git update-index --assum-unchanged my-file.txt
```

- If you didn't create any new files, you can combine `git add` and `git commit` into a single command:

```
git commit -am "Commit message here"
```

- If your latest commit is not published yet, then you can put the current staged changes onto the previous commit.

```
git commit --amend
```

  - *This can also be used to edit an incorrect commit message*
  - If the earlier commit had already been pushed, after amending it you will have to push with `--force` option.

- If you make a commit as the wrong author, you can change it and then amend

```
git config user.name "new name"
git config user.email "email@example.com"
git commit --amend --reset-author
```

- Commit on behalf on someone else

```
git commit -m "message" --author "John Smith <johnsmith@example.com>"
```

- Commit at a specific date

```
git commit -m "Fix memory leak" --date 2018-07-03
```

```
vfs/fatfs: fix stat call failing when called for mount point

FATFS does not support f_stat call for drive root. When handling stat
for drive root, don't call f_stat and just return struct st with S_IFDIR
flag set.

Closes https://github.com/espressif/esp-idf/issues/984
```

- Separate the subject line from body with a blank line
- Limit the subject line to 50 characters
- Do not end the subject line with a period
- Manually wrap each line of the body at 72 characters
- Use the body to explain **what** and **why** instead of **how**

- Show *unstaged* changes on current branch from the commit before it

```
git diff
```

- Show differences for stages files

```
git diff --staged
```

- Show both staged and unstaged changes

```
git diff HEAD
```

- Show changes between two commits

```
git diff 1234abc..6789def # old..new
```

- Show the difference of a file between separate commits

```
git diff 1234abc 6789def myfile.txt
```

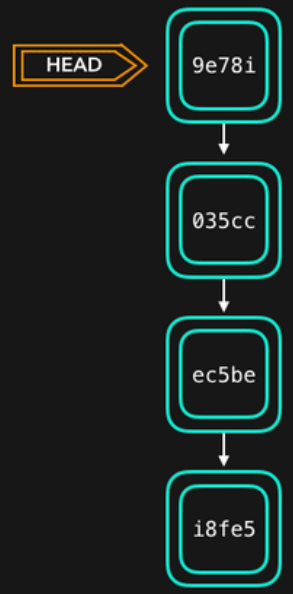
- Produce a patch-compatible diff

```
git diff --no-prefix > some_file.patch
```

Then somewhere else you can reverse it:

```
patch -p0 < some_file.patch
```

Reset 🚨



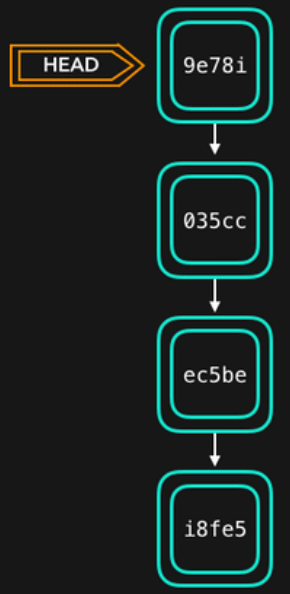
bash

```
master$
```

Git | Soft reset

Points **HEAD** to the specified commit

Keeps changes that have been made
since the new commit that **HEAD** points to, and
keeps the modifications in the working directory



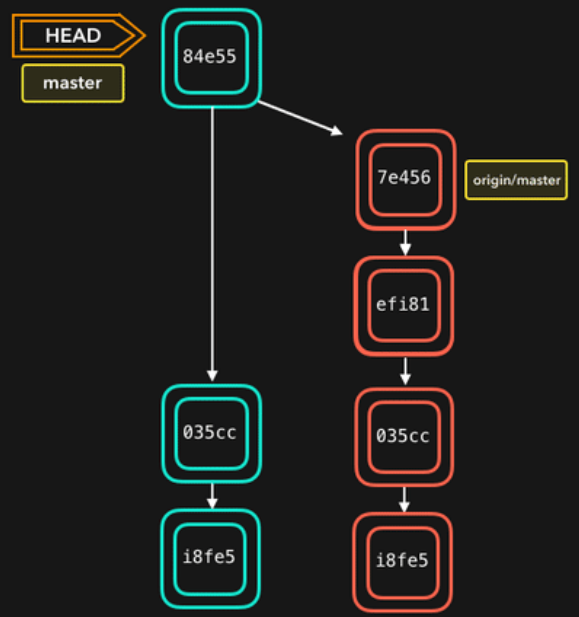
bash

```
master$
```

Git | Hard reset

Points **HEAD** to the specified commit

Discards changes that have been made
since the new commit that **HEAD** points to,
and deletes changes in working directory



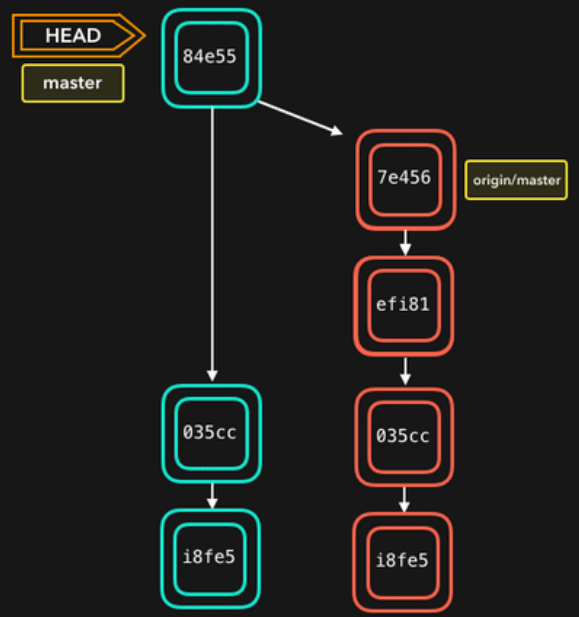
bash

```
master$
```

Git | Reflog

Shows the history of actions in the repo.

With this information, you can easily undo changes that have been made to a repository with **`git reset`**



bash

```
master$ git reflog
84e55 HEAD@{0}: commit(merge)
035cc HEAD@{1}: commit
8d83a HEAD@{2}: reset moving to head~1
18fe5 HEAD@{3}: commit(initial)
```

- Undo changes to a file or directory in the **working copy**

```
git checkout -- file.txt
```

- To **temporarily** jump back to a commit, detach your head

```
git checkout 789abcd
```

This places you at commit 789abcd, you can now make new commits on top this old commit without affecting the branch your head is on.

- Create a new branch, while staying on the current branch

```
git branch <name> [<start-point>]
```

- Switch to an existing branch

```
git checkout <name>
```

- Create a new branch and switch to it

```
git checkout -b <name> [<start-point>]
```

- **<start-point>** can be any revision known to git (e.g. branch name, commit SHA, symbolic reference such as HEAD or a tag name)

- Create a branch from a remote branch

```
git checkout -b <name> <remote_name>/<branch_name>
```

- Listing branches

```
git branch # List local branches
git branch -r # List remote branches
git branch -a # List remote and local branches
```

- Delete a remote branch

```
git push origin --delete <branchName>
```

- Delete a local branch

```
git branch -d <branchName> # Won't delete the branch if it has unmerged changes
git branch -D <branchName> # Delete it event if it has unmerged changes
```

- Quick switch to the previous branch

```
git checkout -
```

- Rename a branch
  - rename the branch you have already checked out

```
git branch -m new_branch_name
```

  - rename another branch

```
git branch -m branch_you_want_rename new_branch_name
```

- List local and remote branches that contain a specific commit

```
git branch -a --contains <commit>
```

- **master branch** 👍
  - Name: `master`
  - Where *integration* happens
  - Produce working code which compiles and passes CI tests
- **feature and bugfix branches** 🐞
  - Name: `feature/xxx` and `bugfix/xxx`
  - Where *development* happens
  - May contain broken or incomplete or testing code
- **release branches** 💪🏽
  - Name: `release/v4.0`
  - Where releases are maintained and bugfixes are backported to

- The `git clone` command is used to copy an existing Git repository from a server to your local machine

```
cd <path where you would like to clone to create a directory>
git clone https://github.com/username/projectname.git
# ssh version of the command
git clone git@github.com:username/projectname.git
```

- If you don't need to have the full history available, you can do a shallow clone

```
git clone [repo_url] --depth 1
```

  - later, if required, you can fetch the rest of the repository

```
git fetch --unshallow
```

- List existing remotes

```
git remote -v
```

- Change Git remote URL

```
git remote set-url remote-name https://github.com/username/repo2.git
```

- Add a new remote repository

```
git remote add github git-repository-url
```

- Push to a remote branch

```
git push <remote_name> <branch_name>
```

- Delete a remote branch

```
git push [remote-name] --delete [branch-name]
```

- Remove local copies of deleted remote branches

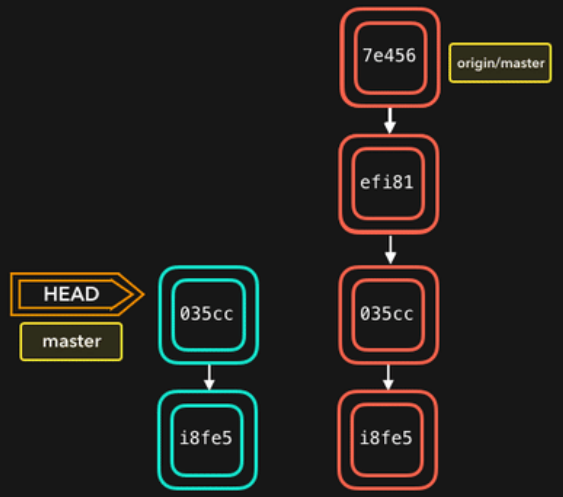
```
git fetch [remote-name] --prune
```

- Updating from upstream repository

```
git fetch remote-name
git merge remote-name/branch-name
```

  - `git pull` combines a `fetch` and a `merge`
  - `git pull --rebase remote-name branch-name` combines a `fetch` and a `rebase`

Fetch VS Pull

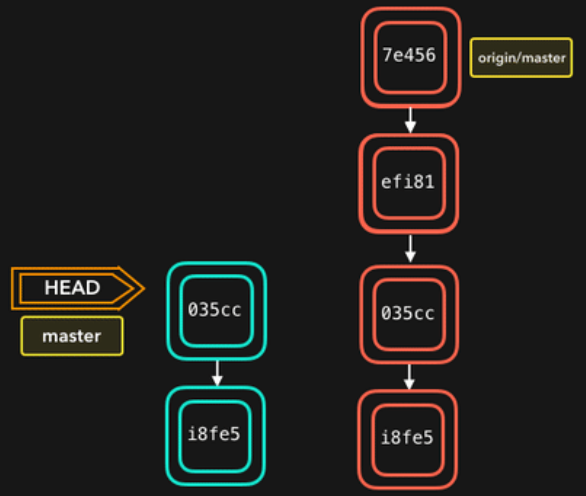


bash

```
master$
```

Git | Fetching

Downloads content from a remote branch or repository without modifying the local state.

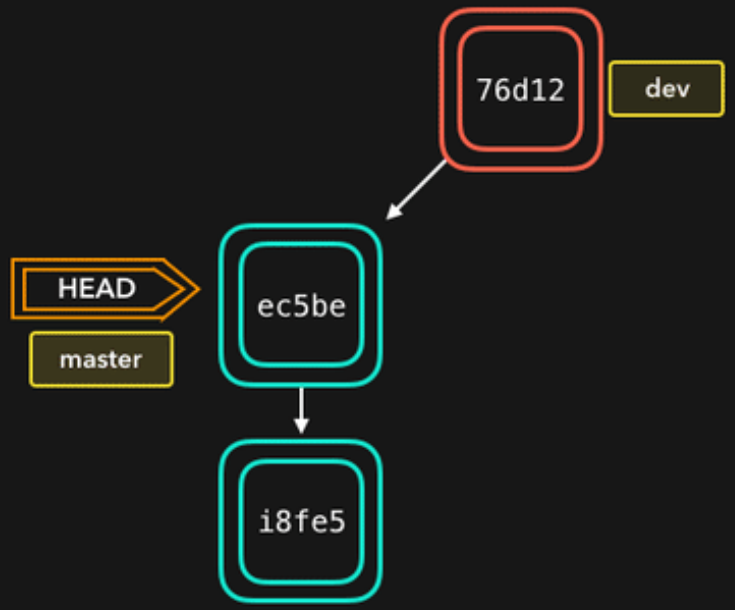


bash

```
master$
```

Git | Pulling

Downloads content from a remote branch/ repository like **`git fetch`** would do, and automatically **`merge`**s the new changes.



bash

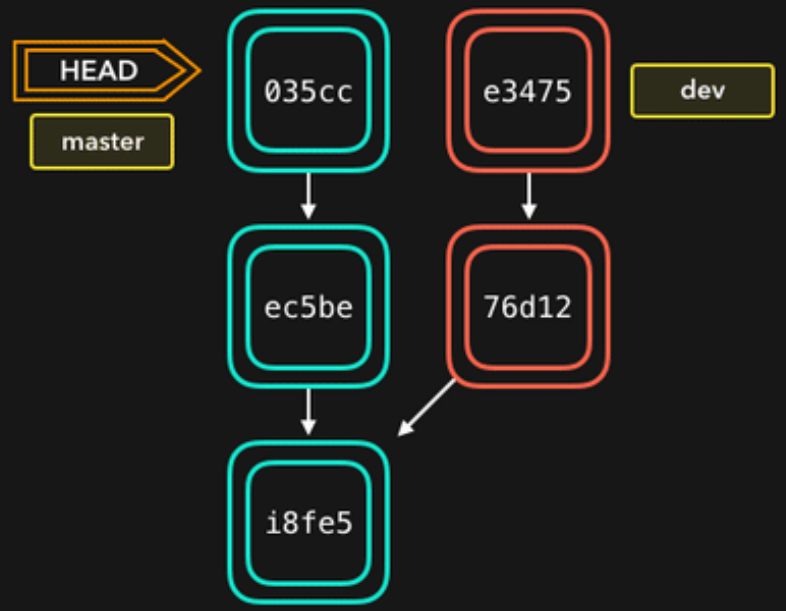
```
master$
```

Git | Merging (fast-forward)

Default behavior when the merging branch has all of the current branch's commits

Doesn't create a new commit, thus doesn't modify existing branches

- No new commit created



bash

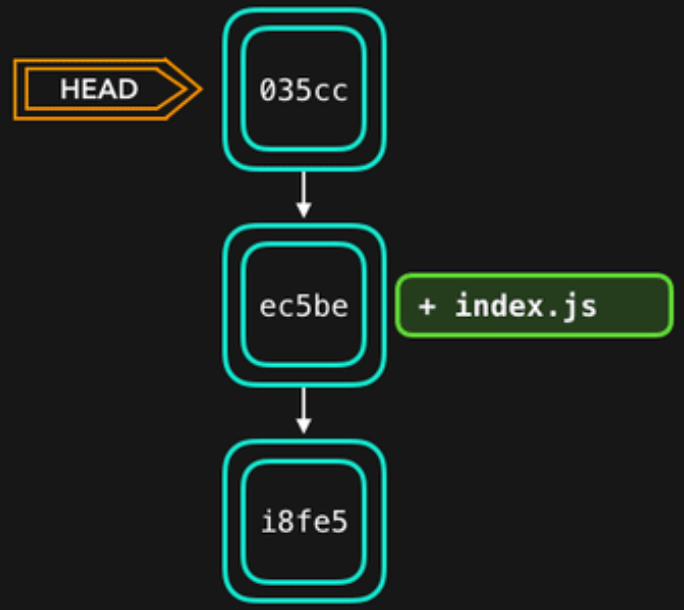
```
master$
```

Git | Merging (no-fast-forward)

Default behavior when current branch contains commits that the merging branch doesn't have

Creates a new commit which merges two branches together without modifying existing branches

- New commit created



bash

```
dev$
```

Git | Reverting

Reverts the changes that commits introduce.
Creates a new commit with the reverted changes.

- Undo a certain commit without modifying history

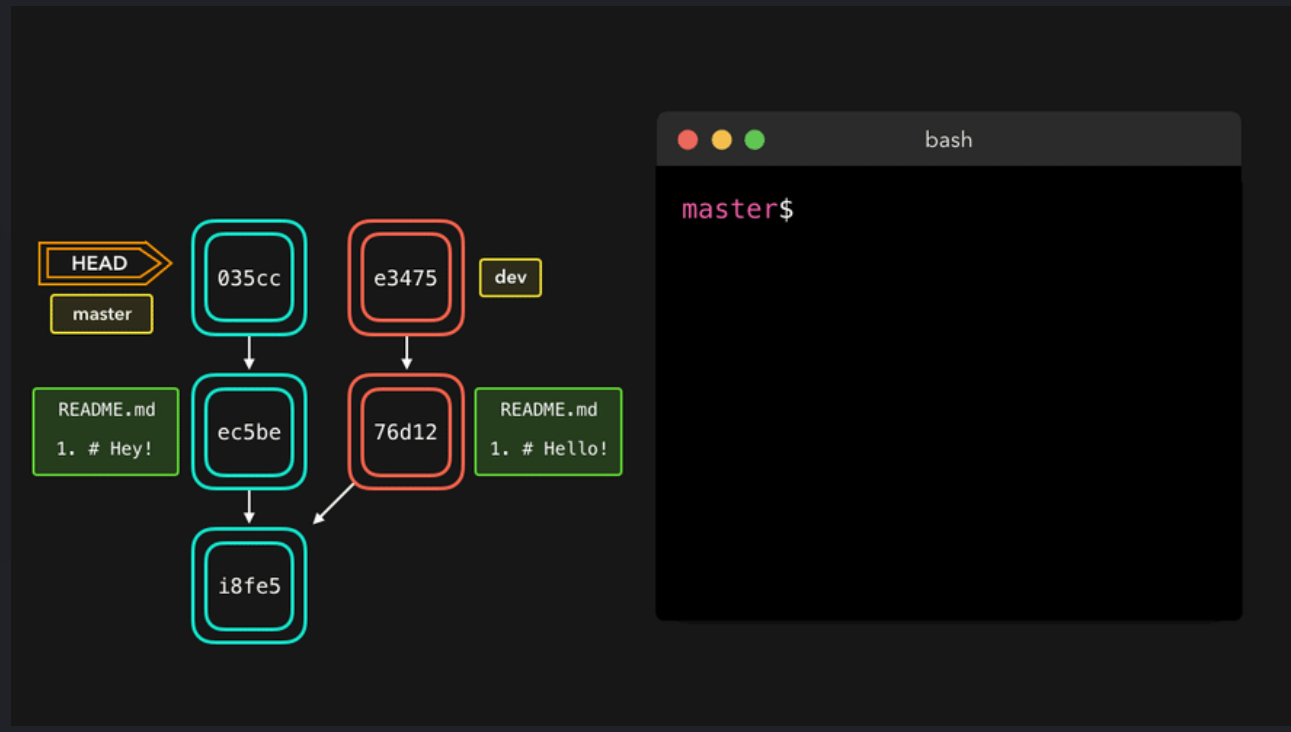
Solve Conflict

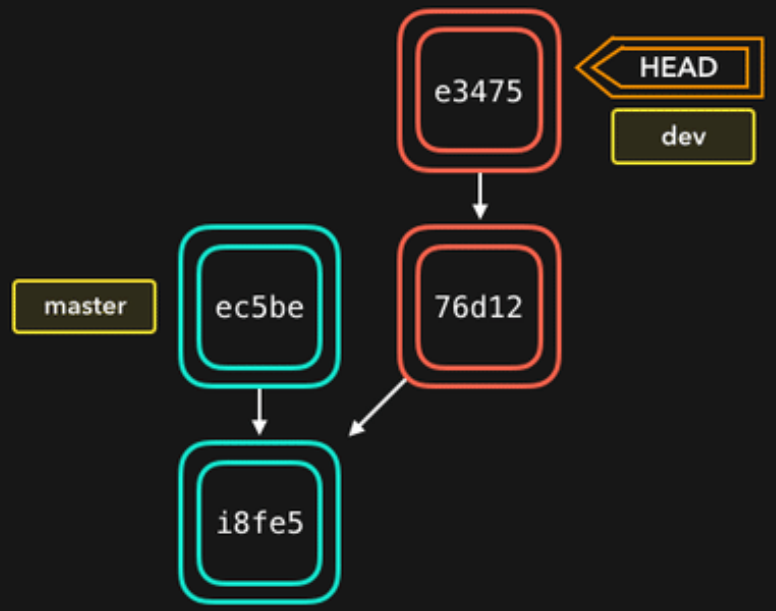
master

```
1 # Hey!
2
3 Welcome to the README of
4 this amazing project
```

dev

```
1 # Hello!
2
3 Welcome to the README of
4 this amazing project
```





bash

```
dev$
```

Git | Rebasing

Copies commits on top of another branch
without creating a commit, which keeps a linear history

Changes the history as new hashes are created
for the copied commits

- It changes the history of the project

- Using interactive rebase, the user can **reword**, **reorder**, **drop**, **split**, **squash** commits
- Rearrange your last three commits

```
git rebase -i HEAD~3
```

  - A file will be opened in your text editor where you will be able to select how your commits will be rebased.
  - After you changed the file, save it and close the editor.
  - This will initiate a rebase with the changes you've applied.
- Aborting an interactive rebase

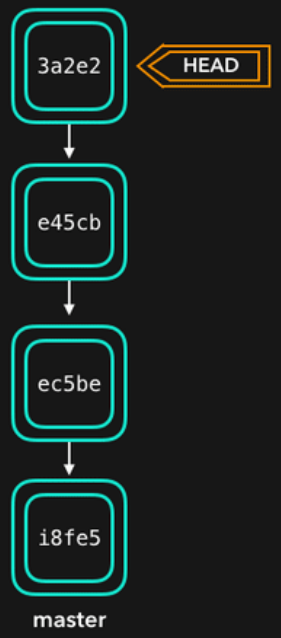
```
git rebase --abort
```

- **git push** complains because your rebase has rewrote the history

```
git push --force-with-lease # this can be solved appending a "force" option
```

- Reorder the commits in the opened text editor

Drop Commits 💧



bash

```
master$
```

Git | Interactive Rebase

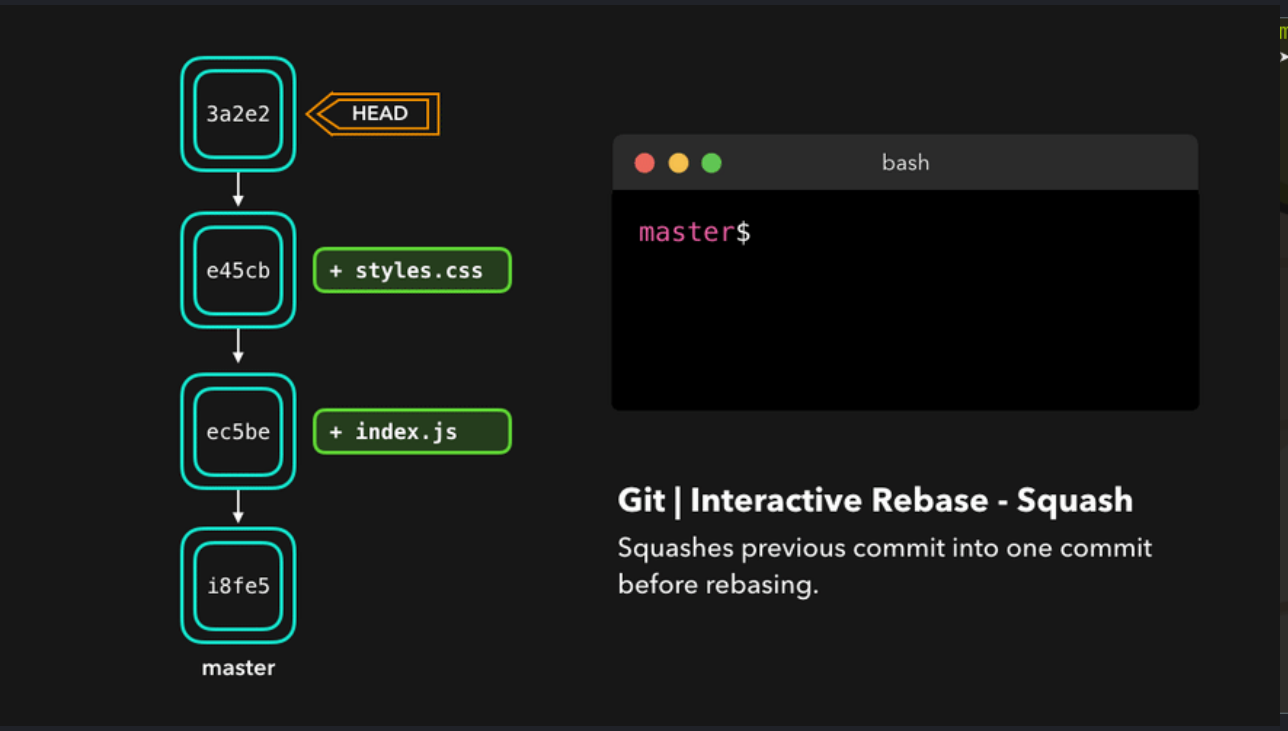
Makes it possible to edit commits before rebasing

Creates new commits for the edited commits/ commits which history has been changed

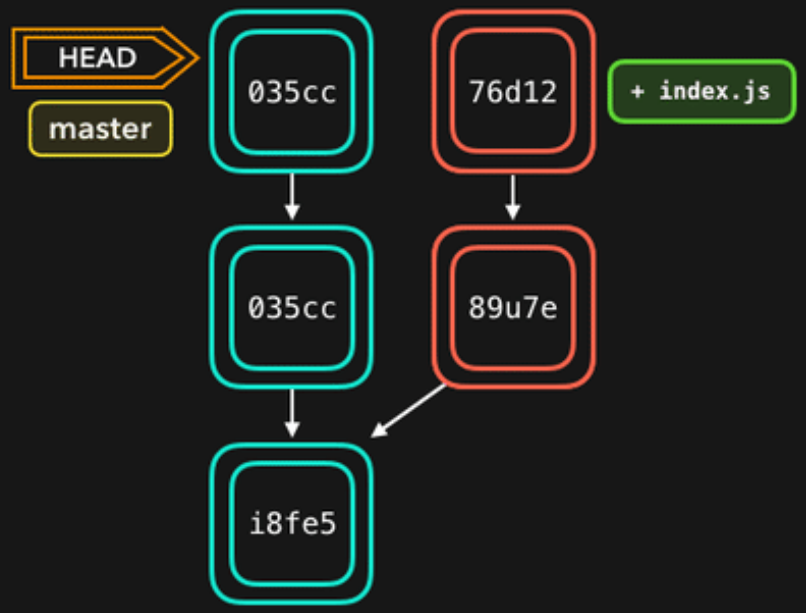
Options: `reword` | `edit` | `squash` | `fixup` | `exec` | `drop`

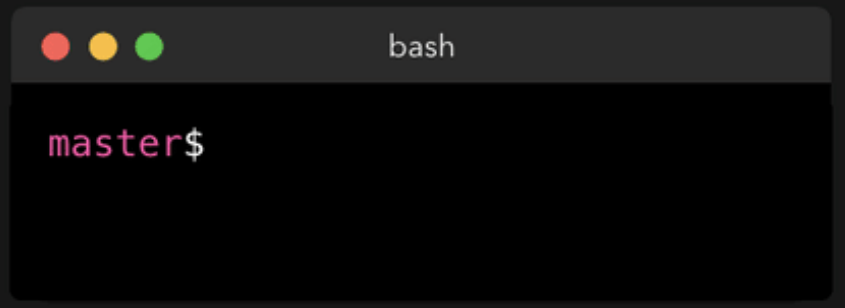
- Replace *pick* with *reword* instead

Squash Commits 📥



- Replace *pick* with *edit* instead



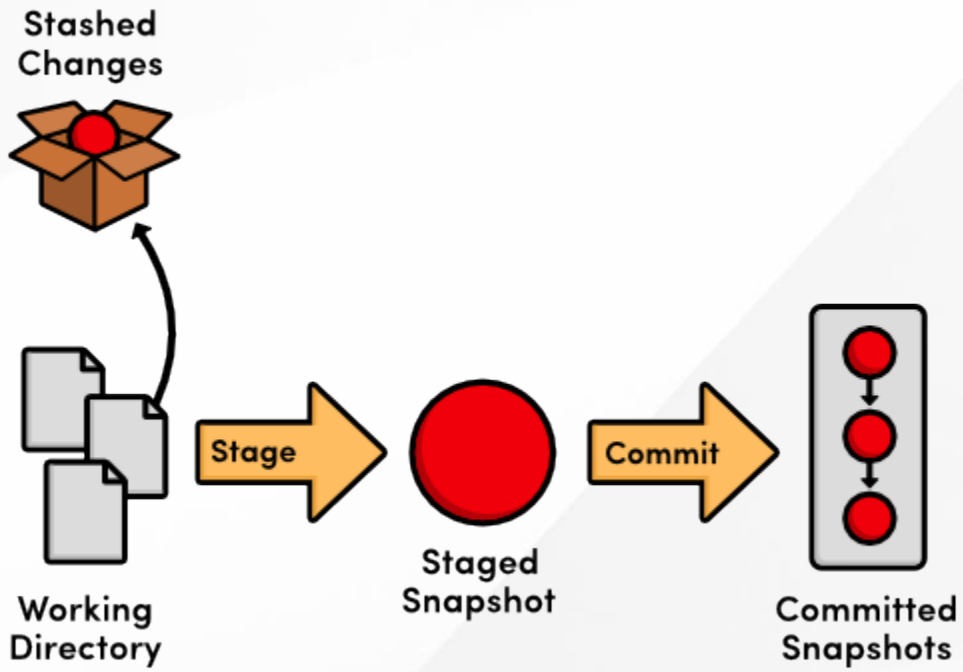


Git | Cherry-picking

Creates a new commit with the changes that the cherry-picked commit introduced.

By default, Git will only apply the changes if the current branch does not have these changes in order to prevent an empty commit.

- The new commit *9e78i* has the same content as *76d12* but a different parent



> When working on a project, you might be **half-way** through a feature branch change when a bug is raised against master. You're **not ready to commit** your code, but you also don't want to lose your changes. This is where **git stash** comes in handy.

- Save the current state of working directory in a stack of stashes

```
git stash
git stash --include-untracked # include all untracked files
git stash save "<whatever message>" # include a message with stash
```

- List saved stashed

```
git stash list
```

- Apply the last stash and remove it from the stack

```
git stash pop
git stash pop stash@{n} # apply specific stash and remove it from stack
```

- Apply the last stash without removing it from the stack

```
git stash apply
```

- Remove stash

```
git stash clear # remove all stash
git stash drop # remove the last stash
```

> To tag specific points in history (e.g. release point) as being important.

- List all available tags

```
git tag
```

- Create a tag on your current branch

```
git tag <tag_name>
```

- Create a tag with some commit

```
git tag tag-name commit-id
```

- Push a commit to remote

```
git push origin tag-name
```

- Cloning a Git repository having submodules

```
git clone --recursive https://github.com/username/repo.git
```

  - This is equivalent to running the following command after the clone is finished

```
git submodules update --init --recursive
```

- Include another Git repository as a folder within your project, tracked by Git:

```
git submodule add https://github.com/ARMmbed/littlefs.git
```

  - you should add and commit `.gitmodules` file, telling Git what submodules should be cloned when `git submodule update` is run

- Moving a submodule

```
git mv /path/to/module new/path/to/module
```

- Removing a submodule (e.g. the_submodule)

```
git submodule deinit the_submodule
git rm the_submodule
```

- To find which commit introduced a bug using a binary search.

```
# start the git bisect
git bisect start
# give a commit where the bug doesn't exist
git bisect good 49c747d
# give a commit where the bug exist
git bisect bad HEAD
```

- Git splits the revision in half and switches the repository to the intermediate revision.
  - Inspect the code to determine if the revision is good or bad:

```
# tell git the revision is good
git bisect good
# if the revision contains the bug, then tell git it's bad
git bisect bad
```

- To abort the bisect process, just issue `git bisect reset`

- Find out who changed a file
  - show the author and commit per line of the specified file

```
git blame test.c
```

  - ignore whitespace-only changes

```
git blame -w test.c
```

  - limits the selection of lines by specified range

```
git blame -L 1,10 test.c
```

  - +offset, -offset

```
git blame -L 108,+30 test.c
```

- Create a patch
  - Make your changes and commit them
  - Convert commits into patch files

```
# convert all commits since <commit-reference> (not including it)
git format-patch <commit-reference>
```

- Apply patches
  - Have the changes from the `.patch` file applied to your current working directory (unstaged)

```
git apply some.patch
```

  - Apply a patch as a commit

```
git am some.patch
git am *.patch # apply all patch files
```

- Clean interactively

```
git clean -i
```

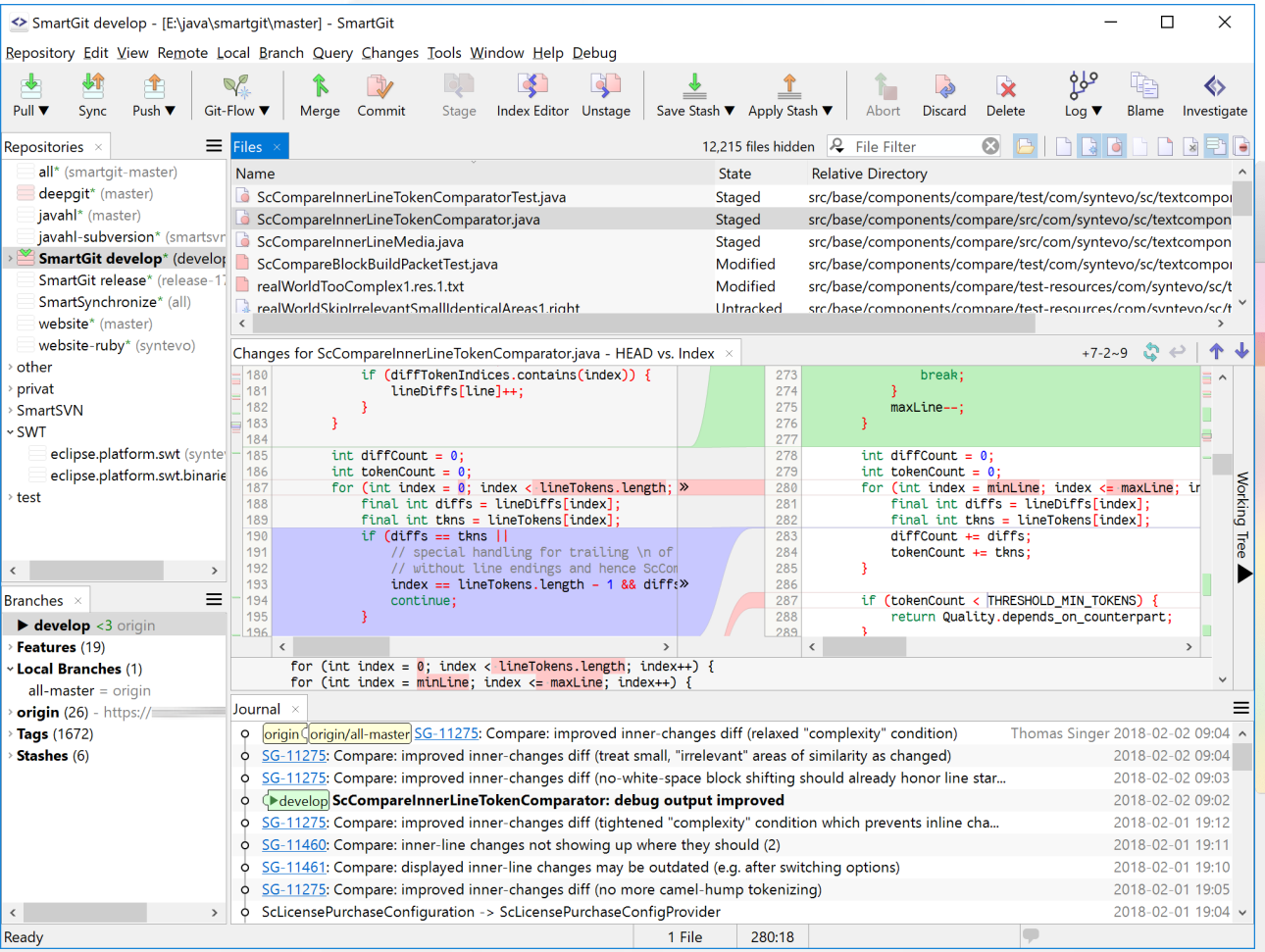
- Forcefully remove untracked files

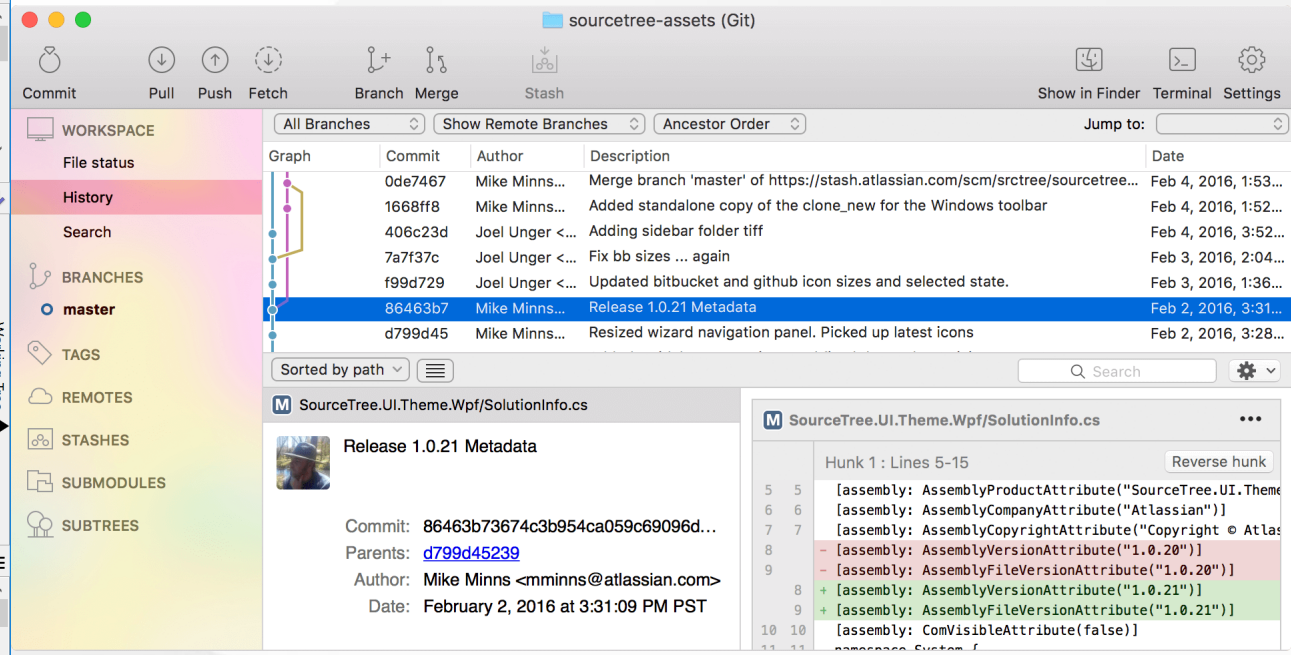
```
git clean -f
```

- Clean ignored files

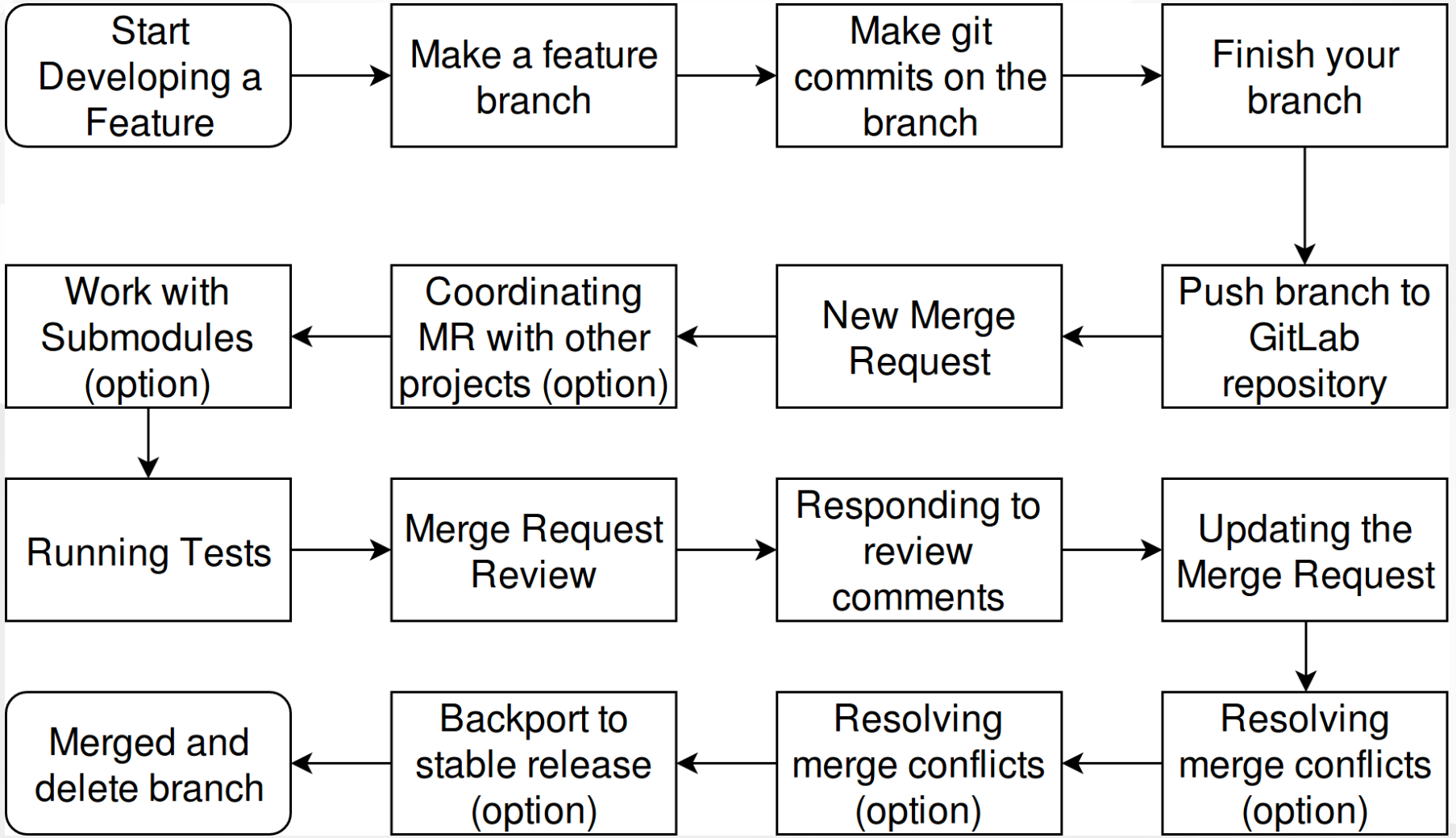
```
git clean -fx
```

Git GUI


SmartGit develop - [E:\java\smartgit\master] - SmartGit
Repository Edit View Remote Local Branch Query Changes Tools Window Help Debug
Pull Sync Push Git-Flow Merge Commit Stage Index Editor Unstage Save Stash Apply Stash Abort Discard Delete Log Blame Investigate
Repositories
Files
12,215 files hidden
File Filter
Name State Relative Directory
ScCompareInnerLineTokenComparatorTest.java Staged
ScCompareInnerLineTokenComparator.java Staged
ScCompareInnerLineMedia.java Staged
ScCompareBlockBuildPacketTest.java Modified
realWorldTooComplex1.res.1.txt Modified
Changes for ScCompareInnerLineTokenComparator.java - HEAD vs. Index
Working Tree
Branches
develop <3 origin
Features (19)
Local Branches (1)
all-master = origin
origin (26)
Tags (1672)
Stashes (6)
Journal
ScCompareInnerLineTokenComparator: debug output improved
Ready
1 File
280:18


sourcetree-assets (Git)
Commit Pull Push Fetch Branch Merge Stash Show in Finder Terminal Settings
WORKSPACE
File status
History
Search
BRANCHES
master
TAGS
REMOTES
STASHES
SUBMODULES
SUBTREES
All Branches
Show Remote Branches
Ancestor Order
Jump to:
Graph Commit Author Description Date
Release 1.0.21 Metadata
Sorted by path
Search
SourceTree.UI.Theme.Wpf/SolutionInfo.cs
Release 1.0.21 Metadata
Commit: 86463b73674c3b954ca059c69096d...
Parents: d799d45239
Author: Mike Minns <mminns@atlassian.com>
Date: February 2, 2016 at 3:31:09 PM PST
Hunk 1 : Lines 5-15
Reverse hunk

Submit a Merge Request on GitLab 🙌


Start Developing a Feature
Make a feature branch
Make git commits on the branch
Finish your branch
Push branch to GitLab repository
New Merge Request
Coordinating MR with other projects (option)
Work with Submodules (option)
Running Tests
Merge Request Review
Responding to review comments
Updating the Merge Request
Resolving merge conflicts (option)
Resolving merge conflicts (option)
Backport to stable release (option)
Merged and delete branch

Submit MR

morris › oh-my-project › Merge Requests › New

New Merge Request

From `feature/add_files` into `master` Change branches

Title add files

Start the title with `WIP:` to prevent a **Work In Progress** merge request from being merged before it's ready.

Add description templates to help your contributors communicate effectively!

Description

Write Preview

Describe the goal of the changes and what reviewers should be aware of.

Markdown and quick actions are supported — Attach a file

Assignee Unassigned — Assign to me

Milestone Milestone

Labels Labels

Merge options

- [x] Delete source branch when merge request is accepted.
- [ ] Squash commits when merge request is accepted.

Submit merge request — Cancel

Commits 1 Changes 3

27 Jun, 2020 1 commit

add files
morris authored 6 minutes ago — be6b3d1b

Good MR Description 💯

espressif › esp-idf › Merge Requests › !9141

打开 Opened 1星期前 by Ivan Grokhotkov

Edit | Close 合并请求

WIP: newlib 3.3.0 _RETARGETABLE_LOCKING support

概览 4 | 提交 3 | 流水线 6 | 变更 10

0/2 话题已解决

1. `_RETARGETABLE_LOCKING` support

Commit `idf/newlib-cygwin@bd547490` implements retargetable locking in newlib.

This MR adapts our previous locking implementation to work when `_RETARGETABLE_LOCKING` is enabled. This feature will be enabled in the future toolchain builds. With the present version of the toolchain, this code doesn't get used. I'm adding it ahead of time to keep fewer changes in the risc-v branch.

When _RETARGETABLE_LOCKING is enabled, newlib locking implementation gets modified as follows:

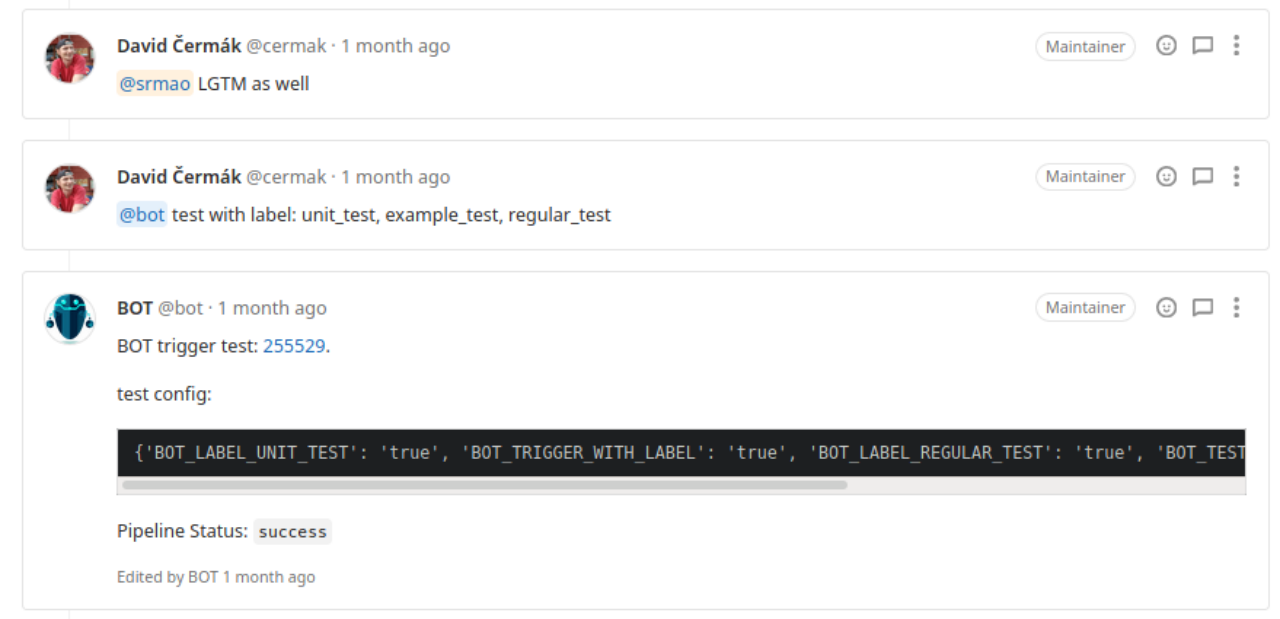
- Legacy ESP-specific _lock_xxx functions are preserved. This is done because ROM copies of newlib in ESP32 and ESP32-S2 rely on these functions through the function pointer table. Also there is some code in IDF which still uses these locking functions.
- New __retarget_lock_xxx functions are introduced. Newlib expects these functions to be provided by the system. These functions work pretty much the same way as the ESP-specific _lock_xxx functions, except one major difference: _lock_acquire receives the lock pointer by value, and as such doesn't support lazy initialization.
- Static locks used by newlib are now explicitly initialized at startup. Since it is unlikely that these static locks are used at the same time, all compatible locks are set to point to the same mutex. This saves a bit of RAM. Note that there are still many locks not initialized statically, in particular those inside FILE structures.

2. `_lock_init` is no longer no-op before the scheduler is started

Newlib _RETARGETABLE_LOCKING implementation does not support lazy initialization by _lock_acquire. Therefore we shouldn't skip calls to _lock_init, including those that happen before the scheduler is started.

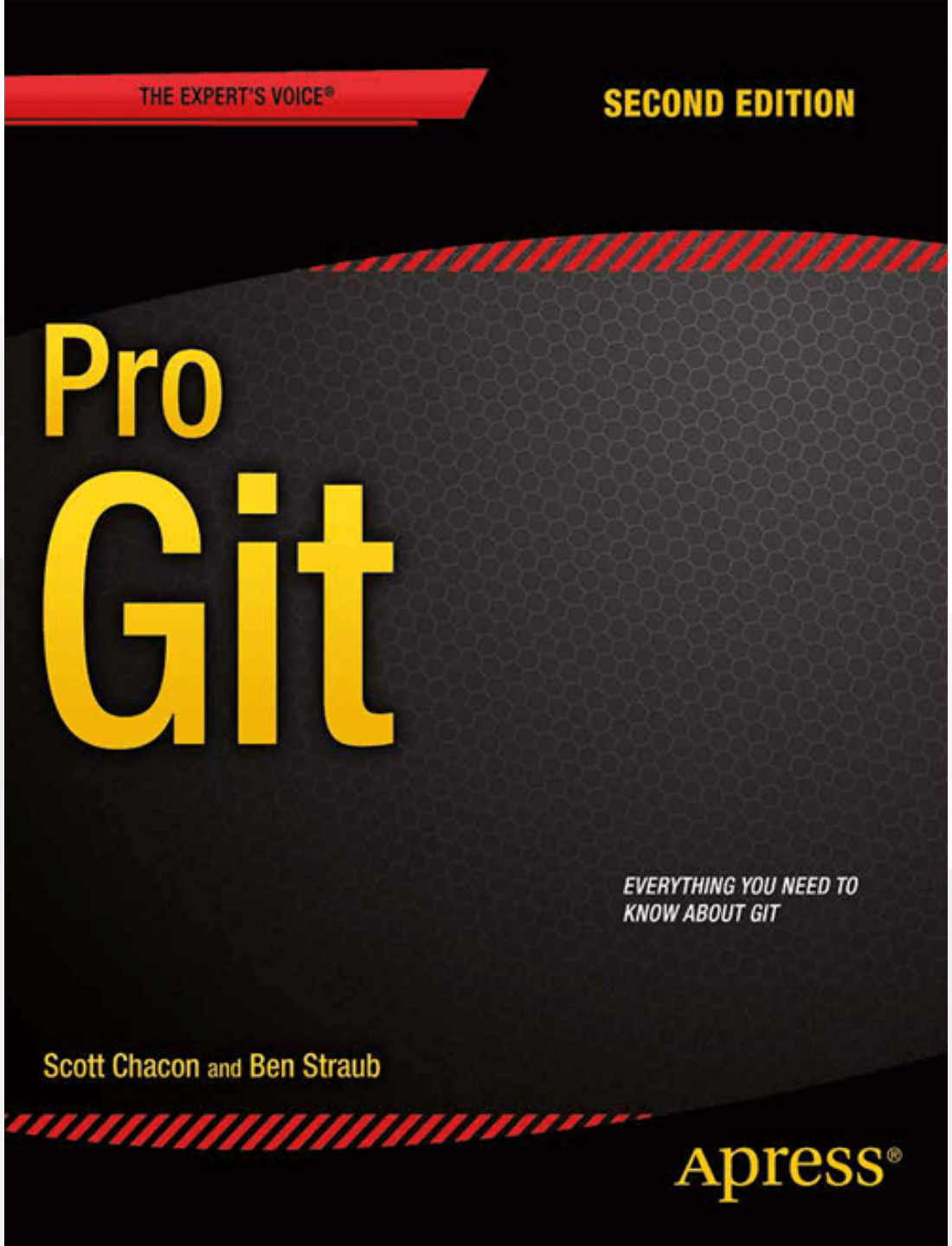
Instead, we check the scheduler state before taking/releasing the lock.

Edited 1星期前 by Ivan Grokhotkov



- Ask @bot to do tests
  - build test
  - unit test
  - integration test
  - language lint
  - ...
- Ask colleagues to review your code
- After you got more than **Two** 👍 and your MR passed CI pipeline, reassign your MR to someone who has the permission to merge

- Be **polite**, even if the code doesn't look good to you
- Try best to **review** PR/MR within 1 week after someone asked you to
- If the MR looks good to you, don't forget to give a **thumbup** 👍


THE EXPERT'S VOICE®
SECOND EDITION
Pro
Git
EVERYTHING YOU NEED TO
KNOW ABOUT GIT
Scott Chacon and Ben Straub
Apress®

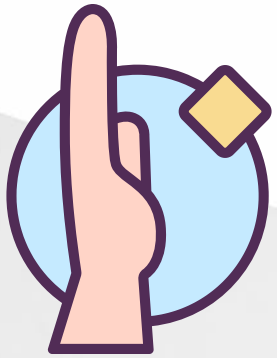
ProGit Book

Learn by Gaming -- GitHug

GitHug Walk Through Guide

Learn in interactive way

Learn Git Branching



- @theavocoder, CS Visualized: Useful Git Commands
- @Stack Overflow Documentation, Git Notes for Professionals