

Ryochan7/DS4Windows

Branch	Updated	Check status	Behind	Ahead	Pull request
<code>master</code>	2 years ago			Default	...
<code>scpvbus_draft</code>	2 years ago		0	72	...
<code>jay</code>	2 years ago		0	0	...
<code>net6_final</code>	2 years ago		114	0	...
<code>net8_testing</code>	2 years ago		80	0	...
<code>joycon_grip</code>	3 years ago		522	0	...
<code>dtotest</code>	3 years ago		532	0	...
<code>absmouse</code>	3 years ago		681	0	...
<code>touchbuttons</code>	3 years ago		746	0	...
<code>deltaaccel_testing</code>	3 years ago		773	0	...
<code>awaitoutputbuffer_test</code>	3 years ago		824	0	...
<code>net6</code>	3 years ago		992	0	...
<code>virtualkbm_refactor</code>	4 years ago		1199	0	...
<code>netframework</code>	4 years ago		1405	7	...
<code>net5</code>	4 years ago		1398	0	...
<code>vigem_1_17</code>	5 years ago		1723	0	...
<code>winforms</code>	6 years ago		2691	6	...
<code>udpserver</code>	7 years ago		3484	0	...

<https://github.com/Ryochan7/DS4Windows>

1. Méthode d'Embranchement

Dans le dépôt DS4Windows, la méthode d'embranchement est généralement basée sur les pratiques courantes de Git :

- Branche Principale : La branche principale est nommée master. C'est la branche stable où le code est prêt pour la production.
- Branches de Fonctionnalité : Les développeurs créent des branches de fonctionnalité pour travailler sur de nouvelles fonctionnalités ou des corrections de bugs. Ces branches sont généralement nommées de manière descriptive, par exemple feature/nouvelle-fonctionnalité.
- Pull Requests (PR) : Une fois qu'une fonctionnalité est terminée, une PR est ouverte pour intégrer les modifications dans la branche principale. Cela permet une révision de code et des discussions avant la fusion.
- Stratégie de Fusion : Les PR peuvent être fusionnées via des méthodes comme le "merge commit" ou le "squash merge", selon les préférences de l'équipe.

2. Gestion des Issues

La gestion des issues dans le dépôt est essentielle pour suivre les bugs, les demandes de fonctionnalités et les améliorations. Voici comment cela fonctionne :

- Création d'Issues : Les utilisateurs et les contributeurs peuvent créer des issues pour signaler des bugs ou proposer des améliorations. Chaque issue est généralement accompagnée d'une description détaillée et, si possible, d'étapes pour reproduire le problème.
- Étiquettes (Labels) : Les issues sont souvent étiquetées pour indiquer leur statut (par exemple, bug, enhancement, help wanted). Cela aide à prioriser et à organiser le travail.
- Assignation : Les issues peuvent être assignées à des membres de l'équipe pour assurer la responsabilité et le suivi.

- Suivi de l'État : Les contributeurs peuvent commenter sur les issues pour fournir des mises à jour ou poser des questions, ce qui favorise la collaboration.

3. Méthode de Management

La méthode de management dans ce dépôt semble être collaborative et ouverte, favorisant l'engagement de la communauté. Voici quelques aspects :

- Contributions Ouvertes : Le dépôt encourage les contributions externes, ce qui est courant dans les projets open source. Les développeurs sont invités à soumettre des PR et à participer aux discussions.
- Documentation : Une bonne documentation est essentielle pour aider les nouveaux contributeurs à comprendre le projet et à s'impliquer. Cela inclut des guides de contribution et des instructions sur la configuration de l'environnement de développement.
- Révisions de Code : Les PR sont examinées par d'autres membres de l'équipe, ce qui permet d'assurer la qualité du code et de partager des connaissances.
- Réunions et Discussions : Bien que cela ne soit pas toujours visible dans le dépôt, des discussions peuvent avoir lieu sur des plateformes externes (comme Discord ou Slack) pour coordonner les efforts de développement.

Conclusion

Le dépôt Ryochan7/DS4Windows utilise une méthode d'embranchement standard avec des branches de fonctionnalité et des PR pour intégrer les modifications. La gestion des issues est bien structurée, permettant un suivi efficace des bugs et des demandes de fonctionnalités. Enfin, la méthode de management est collaborative, favorisant l'engagement de la communauté et la qualité du code

<https://github.com/PCSX2/pcsx2>

PCSX2 - Emulateur de PS2

1. Stratégie de branchement

- **Branche principale** : `master`, visible, sans branche `develop` ou workflow GitFlow. pcsx2.net+15github.com+15github.com+15reddit.com+1github.com+1
- Pas de convention explicite pour les branches secondaires (`feature/`, `fix/`, etc.) observée. Les contributions sont faites via PR ciblant `master`.
- Les releases sont marquées par des tags (ex. `v2.3.xxx`), avec des pré-releases fréquentes. github.com

Limites :

- Risque de conflits directs sur `master`.
 - Absence d'une branche intermédiaire pour stabilisation (ex. `develop` ou `release`).
 - Pas de séparation claire entre dev, tests et prod.
-

2. Gestion des issues et des pull requests

- **Issues** : environ **579 ouvertes**, variées (bugs, améliorations, dépendances). Utilisation de labels (`Bug`, `Enhancement`, `Status: Open`, etc.) et de **3 milestones**. github.com+15github.com+15github.com+15github.comgithub.com+4en.wikipedia.org+4github.com+4github.com+1reddit.com+1
- Les issues sont régulièrement créées (ex. plusieurs dizaines en juin 2025) et bien catégorisées (bug/feature).
- **Pull requests** : **~53 ouvertes**, plus de 7 300 déjà clôturées. Présence de templates et guidelines dans `.github/CONTRIBUTING.md`. github.com+1fr.wikipedia.org+1

Limites :

- Certains PR peuvent stagner sans revue systématique.

- Le nombre élevé d'issues peut rendre la priorisation difficile.
-

3. Intégration continue et versions

- **CI** : GitHub Actions actif, incluant des builds réguliers pour Windows, Linux, macOS, y compris hebdomadaires ("nightly"). [reddit.comreddit.com+1github.com+1](#)
- Les PR déclenchent des workflows, mais on ne sait pas visiblement si les vérifications sont obligatoires avant fusion.
- Politique de versionnement via tags s'appuie sur le canal `v2.3.xxx` pour les nightly pré-releases et `v2.2.0` comme dernière version stable.

Limites :

- Aucun badge ou règle explicite n'indique que le CI doit être réussi pour fusion.
 - Versionnement fonctionnel mais non documenté dans une politique sémantique centralisée.
-

4. Gestion de projet

- L'organisation **GitHub Projects** n'est pas visible publiquement dans le repo.
- **Milestones** limitées à 3, sans backlog ou roadmap claire.
- La coordination semble se faire via Issues, PRs et forums externes (wiki déprécié, forum PCSX2). [pcsx2.net+15github.com+15github.com+15](#)

Limites :

- Pas de tableau Kanban visible.
 - Difficulté à visualiser l'avancement ou la planification.
 - Dépendance à des outils externes pour la communauté.
-

5. Recommandations

1. **Adopter un workflow Git structuré** : ajouter une branche `develop` et utiliser des branches `feature/`, `bugfix/`, `hotfix/`.
 2. **Activer des règles de protection** sur `master` pour imposer CI successful et approbations obligatoires pour les PRs.
 3. **Étendre l'usage des milestones** pour planifier les releases stables et nightly, et lier les PR correspondantes.
 4. **Mettre en place un Project Board** pour organiser les issues par statut (To-do, In progress, Done).
 5. **Documenter la stratégie de versionnement** (semver ou autre) dans `CONTRIBUTING.md` ou `README`.
-

Conclusion

Le projet **PCSX2** est bien mature et organise efficacement ses contributions via :

- CI multi-plateformes,
- un grand volume d'issues bien labellisées et traitées,
- des PR nombreuses et history riche (~7 300 clos). github.comgithub.comreddit.com





























































































Il souffre toutefois d'un manque de **formalisation du workflow Git** (workflow centralisé/fusion directe), d'une documentation distribuée, et d'une **vision globale de gestion** (tableau, roadmap).

En ajoutant davantage de structure au flux Git, aux PRs et à l'organisation (milestones + projet), l'équipe pourrait gagner en **efficacité, qualité et clarté pour les contributeurs**.

Branches

Overview Active Stale All

Q Search branches...

Branch	Updated	Check status	Behind / Ahead	Pull request	
<code>master</code>  	 3 hours ago	 6 / 16	Default		 ...
<code>dependabot/github_actions/ci-deps-fe2ab7588b</code> 	 yesterday	 12 / 22	14 1	 #12892	 ...
<code>dma_gifbackpressure</code> 	 2 days ago	 1 / 11	28 1		 ...
<code>gs_nativegpuclut</code> 	 last month	 12 / 22	143 1	 #12745	 ...
<code>memory-card-destroyer-9000</code> 	 3 months ago	 12 / 22	457 6	 #12187	 ...
<code>gs_shader_cleanup</code> 	 3 months ago	 1 / 11	457 3		 ...
<code>gs_roundspritetest</code> 	 3 months ago	 13 / 23	467 2	 #10278	 ...
<code>gs_roundsprite</code> 	 3 months ago	 13 / 23	467 1	 #6553	 ...
<code>govanify/apprun-wmclass</code> 	 4 months ago	 12 / 22	576 2	 #12328	 ...
<code>mach-exceptions</code> 	 4 months ago	 12 / 22	610 1	 #12280	 ...
<code>gs_channel_coverage_clears</code> 	 5 months ago	 0 / 10	698 1		 ...
<code>chd-cue</code> 	 7 months ago	 13 / 23	798 3	 #12037	 ...
<code>2.2.x</code>  	 7 months ago	 6 / 16	863 0		 ...
<code>2.0.x</code>  	 11 months ago	 6 / 16	1195 0		 ...
<code>gs_wrchacking</code> 	 2 years ago		4289 1		 ...
<code>1.6.x</code>  	 5 years ago		9777 21	 #3242	 ...
<code>1.0</code>  	 7 years ago		16234 36		 ...
<code>1.4.x</code>  	 9 years ago		13219 33	 #2528	 ...
<code>pcsx2-playground-legacy</code>  	 9 years ago		20713 703	 #2282	 ...
<code>1.2.x</code>  	 12 years ago		15677 4		 ...

dolphin-emu/dolphin

1. Stratégie de branches

- Branche principale : **master**.
- Branches secondaires pour les fonctionnalités et correctifs (par ex. **DesHawk** pour TAS, ou dédiées au support Android, build, CI).
- Pas de branche **develop**, mais organisation via tags et sorties continues depuis juillet 2024 (en.wikipedia.org).
- Politique de versionnement en **rolling release** depuis 07/2024, sans numérotation majeure fixe .

Limites :

Pas de zone intermédiaire pour intégrer les changements avant **master**, ce qui peut rendre les merges complexes.

2. Issues et pull requests

- Environ **1 900 issues + PRs** ouvertes (1 870) (opencollective.ecosyste.ms).
- **334 PRs en attente**, certaines en draft, d'autres performantes (journalières à hebdomadaires) .
- Absence de milestones et usage limité des labels (6 existants) .

Limites :

Suivi des contributions laborieux, priorisation floue, Processus de revue potentiellement long.

3. CI/CD et versions

- Utilisation de **GitHub Actions** et de pipelines CI spécialisés (ex. repository **fifoci** pour tests graphiques) (github.com).
- Builds multiplateformes (Windows, macOS, Linux, Android) automatisés.
- Versions labellisées sur rolling release, nightly builds réguliers.
- CI et tests semblent fonctionnels, mais **aucune politique explicite sur gating de PR**.

4. Organisation du projet

- **Pas de GitHub Projects** ou de tableau Kanban.
- Documentation existante : README, Contributing.md, Code of Conduct.
- Site web, wiki, forums et Transifex pour la traduction (ex. wiki, compatibilité, téléchargements) (en.wikipedia.org, github.com, github.com).
- Communication via forums, discussions et plateformes externes.

5. Recommandations

- Ajouter une branche **develop** pour centraliser les ajouts avant **master**.
- Standardiser les noms de branches (**feature/**, **bugfix/**, etc.).
- Introduire labels et milestones pour structurer les issues et PRs.
- Mettre en place un tableau de suivi (**GitHub Projects**) pour visualiser backlog et priorités.
- Appliquer des règles CI obligatoires (tests et revue avant merge).

Conclusion

Le dépôt **dolphin-emu/dolphin** est techniquement performant : CI robuste, builds multiplateformes, contenu riche. Toutefois, l'absence d'organisation formelle (branches dédiées, suivi des contributions, priorisation explicite) peut complexifier la coordination. La mise en place d'outils GitHub pour structurer le flux de contribution renforcerait la collaboration et l'efficacité du projet.

OverviewActiveStateAll

Q Search branches...

Branch	Updated	Check status	Behind	Ahead	Pull request
master	2 days ago			Default	
release-prep-2506a	3 weeks ago		131	2	
release-prep-2506	3 weeks ago		131	0	
release-prep-2503a	2 months ago		742	0	
release-prep-2503	3 months ago		745	0	
release-prep-2412	7 months ago		1087	0	
release-prep-2409	9 months ago		1411	0	
release-prep-2407	last year		1691	0	
stable	10 years ago		26230	39	#9604
4.0-hotfixes	10 years ago		33012	16	#10176

TASEmulators/desmume

1. Stratégie de branchement

- Branche principale unique : `master`.
- Branches secondaires rares, sans convention de nommage claire.
- Tags de versions (ex. `release_0_9_13`) utilisés pour les versions stables.
- Pas de branche de développement (`dev`) ni de gestion structurée des fonctionnalités.

Limites :

Pas de séparation claire entre développement, tests et production. Risques de conflits sur `master`.

2. Gestion des issues et des pull requests

- ~67 issues ouvertes, peu structurées (pas de labels, pas de milestones).
- ~13 pull requests ouvertes, certaines peu suivies.
- Pas de processus clair de relecture ou de validation automatisée.

Limites :

Suivi difficile des priorités. Faible encadrement des contributions.

3. Intégration continue et versions

- GitHub Actions utilisé pour automatiser les builds (Windows, macOS, Linux).
- Présence de versions stables et de nightly builds.
- Pas de vérification automatique imposée sur les PR avant fusion.

Limites :

Processus de validation non systématisé. Politique de versionnement peu explicite.

4. Gestion de projet

- Aucune utilisation de GitHub Projects ou de milestones.
 - Pas de suivi organisé du backlog ou de la feuille de route.
 - Communication communautaire essentiellement externe (forums TAS).
-

5. Recommandations

- Introduire une branche `develop` pour stabiliser les apports avant passage en production.
 - Normaliser les noms de branches (`feature/`, `fix/`, etc.).
 - Utiliser labels et milestones pour organiser les issues.
 - Activer des règles de validation CI sur les pull requests.
 - Mettre en place un tableau de suivi (GitHub Projects) pour clarifier les priorités et l'avancement.
-

Conclusion

Le projet repose sur une structure simple mais gagnerait en efficacité avec une organisation plus formalisée du développement et de la gestion des contributions. Une meilleure utilisation des outils GitHub renforcerait la collaboration et la maintenabilité.

OverviewActiveStaleAll

Q Search branches...

Branch	Updated	Check status	Behind	Ahead	Pull request	
master	5 hours ago	9 / 9			Default	
remove-osmesa	5 hours ago	17 / 17	1	1	#905	
ci-patch-1	2 years ago		208	1	#640	
DesHawk	6 years ago		755	8		
release_0_9_9	6 years ago		2611	10		
rasterizer	8 years ago		1300	1		
powersaves	9 years ago		1523	1		
release_0_9_6	9 years ago		3587	52		
release_0_9_10	9 years ago		2342	16		
release_0_9_11	9 years ago		2096	0		
release_0_9_8	9 years ago		2936	27		
release_0_9_7	9 years ago		3330	21		
release_0_9_5	9 years ago		4011	14		
rerecording_0_9_4_plus	9 years ago		4296	21		
release_0_9_2	9 years ago		5149	14		
release_0_9	9 years ago		5648	17		
rerecording_0_9_2	9 years ago		4965	21		
release_0_9_4	9 years ago		4605	16		
release_0_9_1	9 years ago		5400	9		
release_0_9_3	9 years ago		4682	7		

<https://github.com/miru-project/miru-app>

MIRU APP : APK ANDROID

1. Stratégie de branchement

- Branche principale unique : **dev** (visible sur l'interface), sans branche **main** ou **master** secondaire ni convention claire pour les autres branches.
[github.com+8github.com+8github.com+8](#)
- Pas de branches **feature/**, **fix/**, ni **develop** séparée — tout se fait directement sur **dev** ou via PRs ciblant **dev**.
- Tags de version utilisés pour les releases (v1.8.1, etc.), mais absence d'une branche de pré-production.

Limites : risque de conflits sur **dev**, absence de workflow structuré (pas de **release/** ou **hotfix/**), manque de séparation tests/dev/production.

2. Gestion des issues et des pull requests

- Environ **122 issues** ouvertes, organisées de façon très basique : υπάρχει des labels (? peu visibles), pas de milestones, pas de tri systématique.
- **4–5 pull requests ouvertes**, dont certaines remontant à plusieurs mois, avec peu de suivi ni intégration automatisée.
- Les PR disposent de labels mais ne montrent pas de statut clair de revue ou d'approbation.

Limites : priorisation floue des issues (bugs / fonctionnalités), suivi irrégulier des PRs, absence de revue systématique.

3. Intégration continue et versions

- GitHub Actions actif : builds pour Android, Windows, Web (Flutter). Mention de nightly/stable. [github.com+1github.com+1](#)

- Mais manque de **vérification obligatoire** sur PR : les tests CI ne bloquent pas la fusion.
- Politique de versionnement informelle — tags utilisés, mais pas de stratégie sémantique documentée (`major.minor.patch` ?).

Limites : validation automatique non enforced, versionning pas normé.

4. Gestion de projet

- **Projets GitHub** absents (champ "Projects: 0").
- **Milestones** indisponibles, pas de roadmap ou backlog visible.
[github.com+1reddit.com+1github.com+8github.com+8github.com+8](#)
- Communication se fait via issues, PR, Telegram — mais pas de coordination centralisée.

Limites : planification floue, absence de tableau Kanban ou de roadmap claire.

5. Recommandations

1. **Créer une branche `develop`** pour centraliser les contributions avant release.
 2. **Adopter une convention de branches** (`feature/`, `fix/`, `hotfix/`).
Structurer les issues avec labels (`bug`, `enhancement`, `help wanted`, etc.) et milestones.
Activer les règles de fusion (CI green, approbations).
Mettre en place un Project board (Kanban ou roadmap) pour visualiser l'avancement.
 3. **Documenter la stratégie de versionnement** (sémantique).
-

Conclusion

Le repo **miru-app** dispose de bases solides : build multi-plateformes, releases fréquentes (v1.8.1 en janvier 2024), et une communauté active.

[reddit.com+8github.com+8github.com+8](#)

Cependant, il manque une **structure de gestion de projet formelle**, une **organisation poussée du workflow Git**, et un **processus de revue CI strict**.

Avec une gestion plus formelle des branches, issues, PRs et jalons, la **collaboration et la qualité des contributions** pourraient s’en trouver renforcées.

Branches

OverviewActiveStaleAll

Q Search branches...

Branch	Updated	Check status	Behind	Ahead	Pull request
dev	last year			Default	
new-extension-api	last year	3 / 3	0	12	#255
miru-2	last year		0	0	
feat/linux-support	last year		2	17	#251
fix/update-deps	last year		3	0	
feat/import-local-extension	last year		5	6	#237
feat/new-reader-1	last year		5	27	#225
main	last year		22	0	
feat/extension-api-2	last year		47	0	