L3/S5 – A2022-ITS3111-01 – Rapport git workflow

Étudiant : OUPERE N'SEWA

Tout d'abord, nous devons récupérer une copie du référentiel.

Git svn clone -s -r 40000:HEAD https://svn.parrot.org/parrot # choisissez un engagement récent

-S est pour --stdlayout qui suppose la mise en page recommandée par svn pour les balises, le tronc et les branches.

-R est pour que la révision commence à prendre l'histoire. Si vous voulez inclure toute l'histoire, laissez simplement cette option, mais cela prendra beaucoup de temps, et vous n'avez vraiment pas besoin de tout cela.

REMARQUE : Plus une révision que vous choisissez est ancienne, plus l'importation prendra de temps. Mais vous ne serez pas en mesure de "rester" après la première révision que vous importez. Choisissez judicieusement.

Cela prend un clone du référentiel lors de cette révision ; pour le mettre à jour vers HEAD, vous avez maintenant besoin de :

Git svn rebase

Ce qui est très similaire à svn up.

Du miroir github du leto

Parce que l'importation de toute l'historique prend *très* de temps, DukeLeto a déjà traversé les ennuis et maintient un miroir github de Parrot à l'adresse http://github.com/leto/parrot/tree/upstream .

Git clone git://github.com/leto/parrot.git

Cela créera un répertoire dans le répertoire actuel nommé « perroquet ». Si vous souhaitez spécifier le répertoire à créer, passez un autre argument.

La branche "en amont" est ce qui reflète directement le coffre de Parrot. Pour démarrer une branche locale qui suit la branche en amont :

Cd parrot && git fetch && git checkout -b upstream origin/upstream

Ajout de métadonnées git-svn au clone github

Si vous souhaitez remplir à nouveau les métadonnées SVN, cela peut être fait rapidement en retirant votre clone à jour du dépôt de leto de gihub et en l'ajoutant à votre .git/config

[Svn-à distance "svn"]

Url = https://svn.parrot.org/parrot

Fetch = trunk:refs/remotes/trunk

Ensuite, exécutez cette commande pour trouver le "top commit"

Git show origin/upstream | head -n 1

Et mettez ce hachage de commit dans un fichier .git/refs/remotes/trunk (évidemment remplacer le has par celui de la commande ci-dessus)

Echo c85aaa38b99cedb087e5f6fb69ce6d4a6ac57a0b > .git/refs/remotes/trunk

Et enfin

Git svn fetch

De plein git-svn tarball

Vous pouvez également télécharger une archive tar contenant l'historique complet et toutes les branches ainsi que des métadonnées svn à partir de

Http://technosorcery.net/system/parrot-git-svn.tbz (Mise à jour quotidiennement)

Http://moritz.faui2k3.org/files/parrot-all-git-svn.tar.gz (recommandé ; préparé par Tene, poli par moritz).

Flux de travail local de base

Maintenant, commencez à travailler sur les correctifs. La façon la plus simple de travailler, sans succursales locales, est :

Hack; hack; hack; hack

Git diff # regardez les différences depuis le dernier commit

Si vous souhaitez valider toutes ces modifications, exécutez :

Git commit -a

Et entrez votre message de validation. Cela crée un commit local uniquement. Il n'a pas encore été poussé sur le serveur SVN. Si vous ne souhaitez ajouter que des modifications à partir de certains fichiers, mais pas d'autres, exécutez :

Git ajouter un fichier1 fichier2 fichier3 ...

Git add -i file4 # si vous ne voulez ajouter que quelques diff hunks et pas d'autres

Git commit

Et entrez votre message de validation. Après cela, 'git diff' affichera toujours les modifications non engagées. Pendant que vous ajoutez des modifications à valider, vous pouvez exécuter 'git diff --cached' pour voir ce qui va se passer dans le prochain commit.

Après avoir effectué autant de commits, vous pouvez accéder au serveur svn avec :

Git svn dcommit

Cela mettra également à jour votre arbre local. Pour mettre à jour votre arbre en général, exécutez :

Git svn rebase

Cela mettra à jour votre caisse locale, puis réappliquera vos commits locaux non soumis au-dessus du nouveau coffre.

Si vous voulez obtenir les commits pour toutes les branches qui existent dans votre clone :

Git svn fetch

Astuces soignées

Correctifs entre les changements locaux et le tronc

Si vous voulez voir le jeu de correctifs représentant vos validations et le serveur distant :

Télécommandes/trunk git format-patch

Changez le tronc par le nom de la branche, si vous travaillez contre une branche. Cela créera une série de correctifs numérotés, correspondant à chaque commit que vous avez effectué. Vous pouvez les examiner comme vous le souhaitez.

Rebase interactive

L'une des bonnes choses à propos de git est que vous pouvez modifier vos commits avant de les pousser vers le serveur SVN. Disons que vous avez fait 10 commits sur votre référentiel git local, mais que vous voulez qu'il ressemble à deux lorsqu'il est poussé vers SVN. Tout ce que vous avez à faire est de taper

Git rebase -i HEAD~10

Et git lancera votre $EDITOR et vous permettra d'effectuer ce qu'on appelle une "rebase interactive".

Pick 08464ae bug 9074 Correction d'un mauvais appel de journalisation.

Pick 8750643 bug 9995 Exécutez l'architecture fournie via arch_extract() pour changer 'x64' en 'x86_64' lorsque cela se produit.

Pick af6f7ae bug 9995 Assurez-vous que la chaîne d'architecture se produit sur les limites de mots (c'est-à-dire qu'elle ne fait pas partie d'un mot plus grand)

Choisissez 8e4ef0c Ne générez pas d'avertissement pour les applications génériques.

Pick 1e4d124 Ajouter un module pour gérer les applications personnalisées basées sur des scripts.

Pick 808607d bug 9879 Résoudre les chemins d'accès aux ID d'entité au début d'un processus.

Choisir e02a0db bug 9879 Modifications de mise en forme du code cosmétique.

Pick 3e1cc94 bug 10010 Ajout d'un argument de comparateur facultatif à compare_homes

Pick 218805c bug 10010 Ajout d'un argument de comparateur à l'implémentation par défaut pour plus de cohérence

Pick 21e20fa bug 10010 Made procs_under_home use compare_homes pour gérer l'insensibilité des cas sur les fenêtres

Rebase 05e760e..21e20fa sur 05e760e

#

Commandes :

pick = utiliser commit

edit = utiliser commit, mais arrêter pour modifier

squash = utiliser commit, mais fusion dans le commit précédent

#

Si vous supprimez une ligne ici, CETTE COMMIT SERA PERDUE.

Cependant, si vous supprimez tout, la rebase sera abandonnée.

#

Si vous modifiez l'une des lignes qui commencent par "pick" pour commencer par "squash" à la place, lorsque vous quittez votre $EDITOR, ce commit sera écrasé dans le commit au-dessus et vous serez renvoyé dans $EDITOR pour combiner les messages de validation en un seul. Vous pouvez également écraser plusieurs commits en plus d'un commit dans une seule session ; $EDITOR sera déclenché pour chacun des commits résultants pour que vous puissiez combiner les messages de validation.

Vous pouvez également réorganiser les commits simplement en les déplaçant vers le haut ou vers le bas dans la liste, et vous pouvez supprimer complètement les commits en les supprimant de la liste.

Vous trouverez ci-joint un script perl appelé gsquash qui utilise git log pour rechercher des commits qui n'ont pas encore été poussés vers SVN et construit la commande git rebase -i HEAD~n appropriée.

Git cachette

Disons que vous travaillez sur des choses qui ne sont pas tout à fait prêtes à être enregistrées, mais que vous devez soudainement travailler sur autre chose. Vous pouvez taper git stash, qui enregistrera l'état actuel de votre travail et vous donnera à nouveau un répertoire de travail propre. Vous pouvez ensuite apporter les modifications que vous souhaitez à votre répertoire de travail, les valider, les valider à SVN, puis taper git stash pop pour obtenir ce sur quoi vous travailliez avant de revenir. Vous pouvez également construire une pile de travail caché en apportant à plusieurs reprises des modifications suivies de commandes git stash ; vous pouvez ensuite lister ces modifications cachées avec git stash list. Pour une documentation complète sur le plaisir de git stashing, tapez git stash --help .

Succursale locale

Un autre avantage de git est qu'il est extrêmement facile de créer une branche dans votre référentiel git local :

Git checkout master

Branche git rt1739

Git checkout rt1739

Cela fait d'une succursale locale à l'écart de votre branche principale appelée "rt1739". Vous pouvez valider des modifications à cette branche qui n'affecteront pas votre branche principale, et vous pouvez basculer entre vos branches locales et votre branche principale. En outre, chacune de ces branches est connectée au tronc du référentiel SVN, de sorte que tous les changements engagés à partir de ces branches seront automatiquement fusionnés dans le tronc SVN et apparaîtront dans votre tronc local la prochaine fois que vous exécuterez git svn rebase lorsque vous aurez vérifié votre tronc.

D'autres développeurs peuvent accéder à vos succursales locales, mais pour ce faire, ils devraient pouvoir accéder à votre référentiel git.

Suivi des branches SVN avec GIT

Démarrez la branche svn dans git.

Git svn branch -m "Branche pour le travail sur une fonctionnalité extra cool" extra_cool_feature

Git checkout --track -b extra_cool_feature_local remotes/extra_cool_feature # pour éviter les avertissements de git au sujet de l'ambiguïté

Git reset --hard remotes/extra_cool_feature

... hack/git commit/git commit --amend/git rebase -i HEAD~10/... comme d'habitude

S'engager à svn

Git svn dcommit --dry-run # vérifiez qu'il s'engagera dans la branche svn appropriée.

Git svn dcommit

Fusion du tronc dans la branche

Git checkout master

Git svn rebase

Git checkout extra_cool_branch

Git merge --squash master

Git commit -m "Remper la branche à jour avec trunk"

Fusion de la branche au tronc

Git checkout master

Git svn rebase

Git merge --squash extra_cool_feature

Git commit -m "Fusionner la branche dans le tronc"

Conseils et mises en garde :

N'essayez pas de refonder les modifications déjà engagées. Ils ne vous appartiennent plus.

Exécutez toujours "git svn dcommit --dry-run" avant un commit réel pour vérifier dans quelle branche vos modifications seront validées.

Fusionnez toujours les branches avec "--squash". Sinon, git-svn le fera pour vous et peut échouer épiquement un jour.

Svn n'est pas assez intelligent pour suivre les fusions. Donc, n'essayez pas de fusionner des branches avec un svn pur et un git-svn sur la même branche.

Ignorer les fichiers générés

Les magasins Svn ignorent en tant que métadonnées, git dans les fichiers .gitignore. Vous pouvez ignorer les métadonnées SVN (et les fichiers générés) avec ces fichiers. Tout d'abord, modifiez votre fichier .git/info/exclude pour qu'il contienne :

.Gitignore

... sinon git-svn voudra suivre tous les fichiers .gitignore. Maintenant, générez automatiquement les fichiers .gitignore en exécutant

Git svn create-ignore

Si vous n'avez pas exclu ces fichiers, ils seront mis en place pour votre prochaine validation. Vous ne le voulez probablement pas, car les commettre les ajoutera au SVN de Parrot. Prenez garde!