**Gerenciamento de Configuração - Python**

Lucas Eduardo França – 18286332

Introdução

O Python utiliza um fluxo de trabalho baseado em pull request, ou seja, é criado uma branch no Git, onde é feito as alterações, essas alterações são enviadas para o fork no GitHub e, em seguida, é criada uma pull request no repositório oficial do Python.

Resolvendo Conflitos de Merge

A merge incorpora mudanças dos commits nomeados no branch atual, este comando é usado por git pull para incorporar alterações de outro repositório e pode ser utilizado manualmente para mesclar alterações de um branch em outro. Ao mesclar alterações de ramificações diferentes, é possível que ambas as ramificações possam conter alterações incompatíveis em um ou mais arquivos, se isso ocorrer é preciso resolver manualmente:

1. Verifique os arquivos que apresentam conflitos de mesclagem: git status;
2. Edite os arquivos afetados e leve-os ao estado pretendido;
3. Por fim confirme os arquivos afetados: git add <nome do arquivo>, git merge --continue.

Revisando uma Pull Request

Um dos problemas no processo de desenvolvimento do Python é a falta de revisão do código. Ao navegar pelo rastreador de bugs, é possível ver que vários problemas foram corrigidos, porém não foram incorporados ao repositório principal do código fonte, pois não houve revisão da solução proposta. Deste modo é certo dizer que uma revisão da pull request é tão importante quanto fornecer uma pull request. É importante lembrar que ao realizar uma revisão é crucial que todos os testes sejam feitos antes de ser considerada “pronta para a mesclagem”, e ao invés de apenas aprovar a pull request, deixe comentários que facilitem o entendimento do que foi realizado.

Committing/Rejecting

Quando uma pull request atingir um estado aceitável, ela poderá ser mesclada ou rejeitada. Caso seja rejeitada, é preciso ser realizado alterações até que ela esteja apta a ser mesclada ao código, identificando possíveis causas de problemas. Porém caso a pull request for mesclada, ela poderá estar sendo lançada na próxima versão principal do projeto, ou também poderá estar sendo feito backport para versões mais antigas, como na correção de um bug, caso o desenvolvedor principal acreditar que está fusão é garantida.

Executando e escrevendo testes

Os testes precisam ser completos, rápidos, isolados, repetíveis de forma consistente e o mais simples possível, estes testes são feitos tanto para comportamentos normais ou em condições de erro. Tais testes residem em um diretório (lib/test), onde cada arquivo possui um test\_prefixo.

A diferença entre um teste comum, é que este pode ser confiado a um modulo de suporte feito sob medida para testes em Python, os quais ajudam a suavizar problemas comuns, como diferenças entre plataformas, consumo e limpeza de recursos ou gerenciamento de avisos.

Quando adicionado um teste a um arquivo de teste já existe, é recomendado que seja estudado os outros testes deste arquivo, pois o mesmo ensinará quais precauções devem ser tomadas para tornar os testes robustos e portáteis.

Aceitando Solicitações de Pull Request

Antes de uma pull request ser aceita, é preciso verificar se a mesma está pronta para entrar no código fonte, com algumas perguntas como:

1. Houve a aprovação dos testes?
2. A cobertura do código permaneceu a mesma ou aumentou?
3. As mudanças são aceitáveis?
4. Foram adicionas rótulos apropriados para significar o backport necessário da pull request?
5. A pull request foi feita primeiro contra a branch master?

Após feita essas perguntas é preciso executar todo o conjunto de testes, a fim de garantir que ele seja aprovado antes de mesclar quaisquer alterações de código, também deve ser executado patchcheck para realizar uma verificação rápida quanto as integridades das alterações.

Trabalhando com o Git

Como desenvolvedor principal é preciso tomar certos cuidados ao enviar repositórios, tais como:

1. Não se deve enviar novos branches ao repositório principal, estes ainda pode ser usados no fork para o desenvolvimento de patches, essas branches devem ser enviadas a um repositório separado que passará por uma manutenção antes de ser integrado ao repositório principal.
2. É preciso se comprometer diretamente com a branch master, ou qualquer branch que esteja em manutenção.
3. Para manter o repositório sempre organizado, é fundamental limitar a existência de novas branches para, no máximo, alguns dias.
4. É recomendado manter um fork do repositório principal por perto, pois permite a reversão simples de todas as mudanças locais.

Ciclo de desenvolvimento

As responsabilidade de um desenvolvedor central mudam dependendo da base do tipo da branch, ou em que estágio ela se encontra. No Python a nomenclatura dos lançamentos são dadas como x.y.z (onde “x” são novas versões principais excepcionais, que só ocorrem quando há mudanças fortemente incompatíveis consideradas necessárias e planejadas com antecedência; o “y” são novas versões secundárias são lançamentos de recursos, são lançados anualmente a partir do atual desenvolvimento; o “z” são novas micro versões são lançamentos de correção de bugs, com liberação de aproximadamente 2 meses, são preparados em ramos de manutenção). Também são publicadas versões não finais que recebem qualificadores como: Alpha, Beta, release candidate.

Branches

Há uma branch para cada versão de recursos, seja lançada ou não. A branch master é a branch mais próxima do lançamento. Em algum momento do ciclo de vida de uma nova versão, é criado uma branch de manutenção para hospedar todas as atividades de correção de bugs para outras micro versões em uma versão de recurso.

Em Python existem as branches de manutenção, as quais são mantidas após o lançamento de uma nova release, estas são usadas para correções de bugs e sua compatibilidade não deve ser quebrada em nenhum ponto entre versões secundárias irmãs (3.5.1, 3.5.2...). Após uma nova branch de manutenção ser criada, a branch de manutenção antiga entra em modo de segurança. Uma branch com mais de 5 anos, fora do modo de manutenção, é uma branch de segurança, as únicas alterações que podem ocorrer nelas são as que corrigem problemas exploráveis por invasores, como travamentos, aumento de privilégios etc., quaisquer outras alterações não devem ser tratadas em uma branch de segurança.

Quando uma versão final está sendo finalizada, apenas o gerenciador da versão pode realizar as alterações na branch, após sua publicação, o ciclo de desenvolvimento se inicia para a próxima versão.

Integração contínua

O Python apresenta um conjunto de máquinas dedicadas, chamadas buildbots, usadas para integração contínua, as quais afirmam que não há regressões nas branches de desenvolvimento e manutenção, eles abrangem uma série de combinações de hardware/sistema operacional. Suas etapas consiste em verificar a árvore de origem, copilá-lo, executar os códigos de testes e limpar a árvore de construção.

Porém é de responsabilidade do desenvolvedor principal, verificar os resultados da compilação automática depois de enviar uma alteração ao repositório.

Referências: <<https://devguide.python.org/>> <https://git-scm.com/docs/>