PYTHON

Estratégia de branching:

Crie uma nova filial onde seu trabalho para o problema irá, por exemplo:
`git checkout -b fix-issue-12345 master`

Se ainda não houver um problema, crie-o. Problemas triviais (por exemplo, correções de erros de digitação) não exigem a criação de nenhum problema.

Empurre o branch em seu fork no GitHub e crie uma solicitação pull . Inclua o número do problema usando `bpo-NNNNna` descrição da solicitação de pull. Por exemplo:
`bpo-12345: Fix some bug in spam module`

Status dos branches Python

Ramo	Cronograma	Status	Primeiro lançamento	Fim da vida	Gerente de liberação
mestre	PEP 619	recursos	*2021-10-04*	*TBD*	Pablo Galindo Salgado
3,9	PEP 596	correção de bug	*05/10/2020*	*TBD*	Łukasz Langa
3,8	PEP 569	correção de bug	14/10/2019	*2024-10*	Łukasz Langa
3,7	PEP 537	segurança	27/06/2018	*2023-06-27*	Ned Deily
3,6	PEP 494	segurança	23-12-2016	*2021-12-23*	Ned Deily
3,5	PEP 478	segurança	13/09/2015	*13-09-2020*	Larry Hastings

O branch master é atualmente o futuro Python 3.10, e é o único branch que aceita novos recursos.

Status:

recursos:	novos recursos, correções de bugs e correções de segurança são aceitos.
pré-lançamento:	correções de recursos, correções de bugs e correções de segurança são aceitas para o próximo lançamento de recursos.
bugfix:	correções de bugs e correções de segurança são aceitas, novos binários ainda são lançados. (Também chamado de modo de **manutenção** ou versão **estável**)
segurança:	apenas correções de segurança são aceitas e não há mais binários lançados, mas novas versões apenas de origem podem ser lançadas
fim da vida:	o ciclo de liberação está congelado; nenhuma outra alteração pode ser enviada a ele.

As datas em *itálico* são programadas e podem ser ajustadas.
Por padrão, o fim da vida útil é agendado 5 anos após o primeiro lançamento, mas pode ser ajustado pelo gerente de lançamento de cada filial. Todas as versões do Python 2 atingiram o fim de sua vida útil.

Merging

Resolvendo conflitos de mesclagem

Ao mesclar alterações de ramificações diferentes (ou variantes de uma ramificação em repositórios diferentes), as duas ramificações podem conter alterações incompatíveis em um ou mais arquivos. Eles são chamados de “conflitos de mesclagem” e precisam ser resolvidos manualmente da seguinte forma:

1. Verifique quais arquivos têm conflitos de mesclagem:
2. `git status`
3. Edite os arquivos afetados e traga-os ao estado final pretendido. Certifique-se de remover os “marcadores de conflito” especiais inseridos pelo git.
4. Confirme os arquivos afetados:
5. `git add <filenames> git merge --continue`

Ao executar o comando final, git pode abrir um editor para escrever uma mensagem de confirmação. Geralmente, não há problema em deixar como está e fechar o editor

Como revisar uma solicitação pull request

Um dos gargalos no processo de desenvolvimento Python é a falta de revisões de código. Se você navegar no rastreador de bug, verá que vários problemas foram corrigidos, mas não podem ser incorporados ao repositório principal de código-fonte, porque ninguém revisou a solução proposta. A revisão de uma solicitação pull pode ser tão informativa quanto fornecer uma solicitação pull e permitirá que você faça comentários construtivos sobre o trabalho de outro desenvolvedor. Este guia fornece uma lista de verificação para o envio de uma revisão de código. É um equívoco comum pensar que, para ser útil, uma revisão de código deve ser perfeita. Este não é o caso! É útil apenas testar a solicitação pull e / ou brincar com o código e deixar comentários na solicitação pull ou rastreador de problemas.

6. Se você ainda não fez isso, obtenha uma cópia do repositório CPython seguindo o guia de configuração , construa-o e execute os testes.
7. Verifique o rastreador de bug para ver quais etapas são necessárias para reproduzir o problema e confirme se você pode reproduzir o problema em sua versão do REPL do Python (o prompt de shell interativo), que pode ser iniciado executando /python dentro do repositório.
8. Faça check-out e aplique a solicitação pull
9. Se as alterações afetarem qualquer arquivo C, execute o build novamente.
10. Inicie o Python REPL (o prompt do shell interativo) e verifique se você pode reproduzir o problema. Agora que a solicitação de pull foi aplicada, o problema deve ser corrigido (em teoria, mas erros acontecem! Uma boa revisão visa detectá-los antes que o código seja mesclado no repositório Python). Você também deve tentar ver se há algum caso secundário neste ou em problemas relacionados que o autor da correção possa ter esquecido.

11. Se você tiver tempo, execute todo o conjunto de testes. Se você está sem tempo, execute os testes para o (s) módulo (s) onde as alterações foram aplicadas. No entanto, esteja ciente de que, se estiver recomendando uma solicitação pull como 'pronta para mesclagem', você deve sempre garantir que todo o conjunto de testes seja aprovado.

Comprometendo / rejeitando

Assim que sua solicitação pull atingir um estado aceitável (e, portanto, considerada "aceita"), ela será mesclada ou rejeitada. Se for rejeitado, não leve para o lado pessoal! Seu trabalho ainda é apreciado, independentemente de sua solicitação pull ser mesclada. Equilibrar o *que* entra e o *que não* entra em Python é complicado e simplesmente não podemos aceitar as contribuições de todos.
Mas se sua solicitação pull for mesclada, ela irá para o VCS do Python para ser lançada com a próxima versão principal do Python. Também pode ser feito backport para versões mais antigas do Python como uma correção de bug se o desenvolvedor principal que está fazendo a fusão acreditar que isso é garantido.

Executando e escrevendo testes

Executando

A maneira mais curta e simples de executar o conjunto de testes é o seguinte comando do diretório raiz do seu checkout (depois de criar o Python):

```
./python -m test
```

Você pode precisar alterar este comando da seguinte maneira em toda esta seção. Na maioria dos sistemas Mac OS X, substitua `./python` por `./python.exe`. No Windows, use `python.bat`. Se estiver usando Python 2.7, substitua `test`por `test.regrtest`.

Se você não tem acesso fácil a uma linha de comando, pode executar o conjunto de testes a partir de um shell Python ou IDLE:

```
>>>
>>> from test import autotest
```

Isso executará a maioria dos testes, mas excluirá uma pequena parte deles; esses testes excluídos usam tipos especiais de recursos: por exemplo, acessar a Internet ou tentar reproduzir um som ou exibir uma interface gráfica em sua área de trabalho. Eles são desabilitados por padrão para que a execução do conjunto de testes não seja muito intrusiva. Para habilitar alguns desses testes adicionais (e para outras sinalizações que podem ajudar a depurar vários problemas, como vazamentos de referência), leia o texto de ajuda:

```
./python -m test -h
```

Se você deseja executar um único arquivo de teste, simplesmente especifique o nome do arquivo de teste (sem a extensão) como um argumento. Você provavelmente também deseja ativar o modo verboso (usando `-v`), para que as falhas individuais sejam detalhadas:

```
./python -m test -v test_abc
```

Para executar um único caso de teste, use o `unittest` módulo, fornecendo o caminho de importação para o caso de teste:

```
./python -m unittest -v test.test_abc.TestABC
```

Se você tiver uma máquina com vários núcleos ou várias CPU, pode ativar o teste paralelo usando vários processos Python para acelerar as coisas:

```
./python -m test -j0
```

Se você estiver executando uma versão do Python anterior a 3.3, deverá especificar o número de processos a serem executados simultaneamente (por exemplo `-j2`).

Por fim, se você deseja executar testes em um conjunto de configurações mais árduo, pode executar `test` como:

```
./python -bb -E -Wd -m test -r -w -uall
```

Os vários sinalizadores extras passados para Python tornam-no muito mais rígido sobre várias coisas (o `-Wd` sinalizador deve estar em algum ponto, mas o conjunto de testes não atingiu um ponto onde todos os avisos foram tratados e, portanto, não podemos garantir que um bug o Python gratuito completará adequadamente uma execução de teste com). O sinalizador para o executor de teste faz com que ele execute os testes em uma ordem mais aleatória, o que ajuda a verificar se os vários testes não interferem uns com os outros. O sinalizador faz com que os testes com falha sejam executados novamente para ver se as falhas são temporárias ou consistentes. O flag permite a utilização de todos os recursos disponíveis para não saltar testes que requeiram, por exemplo, acesso à Internet.`-W error-W error-r-w-uall`

Para verificar se há vazamentos de referência (necessário apenas se você modificou o código C), use o `-R` sinalizador. Por exemplo, primeiro executará o teste 3 vezes para estabelecer a contagem de referência e, em seguida, executará mais 2 vezes para verificar se há vazamentos.`-R 3:2`

Você também pode executar o `Tools/scripts/run_tests.py`script conforme encontrado em um checkout CPython. O script tenta equilibrar velocidade com meticulosidade. Mas se você deseja os testes mais completos, deve usar a abordagem extenuante mostrada acima.

Escrevendo testes

Escrever testes para Python é muito parecido com escrever testes para seu próprio código. Os testes precisam ser completos, rápidos, isolados, repetíveis de forma consistente e o mais simples possível. Tentamos fazer testes para comportamento normal e para condições de erro. Os testes residem no `Lib/test` diretório, onde cada arquivo que inclui testes possui um `test_` prefixo.

Uma diferença com o teste comum é que você é incentivado a confiar no `test.support` módulo. Ele contém vários auxiliares que são feitos sob medida para o conjunto de testes do Python e ajudam a suavizar problemas comuns, como diferenças de plataforma, consumo e limpeza de recursos ou gerenciamento de avisos. Esse módulo não é adequado para uso fora da biblioteca padrão.

Ao adicionar testes a um arquivo de teste existente, também é recomendável estudar os outros testes desse arquivo; ele ensinará quais precauções você deve tomar para tornar seus testes robustos e portáteis.

Release

Um branch se preparando para um lançamento RC só pode ter correções de bugs aplicadas que foram revisadas por outros desenvolvedores principais. Geralmente, esses problemas devem ser graves o suficiente (por exemplo, travamentos) para que mereçam ser corrigidos antes do lançamento final. Todas as outras questões devem ser adiadas para o próximo ciclo de desenvolvimento, uma vez que a estabilidade é a maior preocupação neste momento.

Você **não pode** pular a revisão por pares durante uma RC, não importa quão pequena! Mesmo que seja uma alteração simples de copiar e colar, **tudo** requer revisão por pares de um desenvolvedor principal.

Final

Quando uma versão final está sendo cortada, apenas o gerenciador de versão (RM) pode fazer alterações no branch. Depois que a versão final é publicada, o ciclo completo de desenvolvimento começa novamente para a próxima versão secundária.

Versões

As versões conhecidas do Python que o problema afeta e para as quais deve ser corrigido.

Portanto, se um problema para um novo recurso for atribuído, por exemplo, Python 3.8, mas não for aplicado antes do lançamento do Python 3.8.0, este campo deve ser atualizado para dizer Python 3.9 como a versão e eliminar o Python 3.8.

Prioridade

Qual é a gravidade e urgência?

Prioridade	**Descrição**
baixo	Isso é para insetos de baixo impacto.
normal	O valor padrão para a maioria dos problemas arquivados.
Alto	Tente corrigir o problema antes da próxima versão final.
crítico	Definitivamente deve ser consertado para o próximo lançamento final.
bloqueador diferido	O problema não vai atrasar o próximo lançamento, *n* . Ele será promovido a um *bloqueador de lançamento* para o lançamento seguinte, *n + 1* .
bloqueador de liberação	O problema **deve** ser corrigido antes que *qualquer* lançamento seja feito, por exemplo, bloqueará o próximo lançamento mesmo que seja um lançamento alfa.

Como diretriz, *crítico* e acima são geralmente reservados para travamentos, regressões graves ou quebra de APIs muito importantes. Se um bug é um *bloqueador de lançamento* para a programação de lançamento atual, é decidido pelo gerente de lançamento. Os triagers podem recomendar essa prioridade e devem adicionar o gerenciador de lançamento à *lista intrometida* . Se necessário, consulte o cronograma de lançamento e o PEP associado ao lançamento para obter o nome do gerente de lançamento.