Tails - Feature #15287

Make it possible to reproducibly generate IUKs in CI

02/05/2018 04:13 PM - anonym

Status: Resolved
Priority: High
Assignee: 
Category: Continuous Integration
Target version: Tails_4.2
Feature Branch: feature/15281-single-squashfs-diff
Type of work: Code

Description

Since we'll generate a lot more of IUKs each release uploading them will be painful for RMs with slow internet connections.

For example, the following VM would do:

- 4 virtual CPUs (so the IUK generation won't take too long)
- 1 GB of RAM (I used to generate IUKs in such a VM two years ago but YMMV)
- 10 GB /tmp for tails-iuk's needs, and where we store the generated IUKs (before uploading them to rsync.lizard)
- Access to Jenkins build artifacts

Related issues:

Related to Tails - Feature #17262: Make the build of overlayfs-based IUKs rep... Resolved
Related to Tails - Bug #17361: Streamline our release process Confirmed
Related to Tails - Feature #15281: Stack one single SquashFS diff when upgrading Resolved 04/13/2016
Related to Tails - Feature #17385: Grow /home on rsync.lizard Resolved
Related to Tails - Feature #17412: Drop the need for dedicated temporary stor... Resolved
Blocks Tails - Feature #16052: Document post-release reproducibility verifica... Confirmed 10/15/2018
Blocks Tails - Feature #16209: Core work: Foundations Team

Associated revisions

Revision 150f1638 - 12/26/2019 10:34 AM - intrigeri

IUK creation: delete temporary directory on success (refs: #15287)

tails-create-iuk previously left temporary files behind. In particular, even on success, the "squashfs_src" directory would remain on the filesystem.

It's no big deal in general, but our build_IUKs Jenkins job runs tails-create-iuk as root with sudo, so the leftover temporary files are owned by root, and then the "workspace-cleanup" step (itself run as the "jenkins" user) can't delete them, which causes 2 problems:

- These temporary files would endlessly accumulate on isobuilders, without any mechanism to clean them up automatically.

- Confusing output in the Jenkins console:

```
cannot remove path when cwd is /var/lib/jenkins/workspace/build_IUKs/j4gMvgS3WH for /var/lib/jenkins/workspace/build_IUKs/j4gMvgS3WH: at /usr/share/perl/5.24/File/Temp.pm line 1583.
Archiving artifacts
[WS-CLEANUP] Deleting project workspace...
Cannot delete workspace: null
```

Option not to fail the build is turned on, so let's continue
This problem has not happened since a few years, mostly because I'm now aware of it and thus careful. Moreover, we now have a build_IUKs Jenkins job, that takes a TAILS_GIT_COMMIT parameter, and can thus be used to test new IUK generation code before it’s merged and affects a RM, on real-world scenarios. I'm using this regularly when working on the IUK codebase. As it happens, this job runs on Jenkins' isobuilders, that currently run Stretch.

So all things considered, I'm now confident that this note does more harm (scaring the RM) than good (providing useful and actionable information).

Our isobuilders should be fine then.

- Access to Jenkins build artifacts

Our isobuilders have access to the ISO archive so the easiest way is to push the new ISO there before generating the IUKs.

There's no chance I do all this work myself during the 3.6 cycle. I could deploy a new Jenkins job if someone else write the code that will build the needed IUKs and tells me how this job should behave (input, output, how it'll be triggered).
intrigeri wrote:

I could deploy a new Jenkins job [...] 

Is Jenkins the right tool here? As a RM, I see no benefit. To me, the ideal would be that all RMs got access to some VM on Lizard fulfilling the above criteria, and that I write some shell script in the release process that simply is copy-pasted into a terminal to do the IUK building in said VM over SSH. This way I can much more easily debug and find workarounds if there's problems.

intrigeri wrote:

I could deploy a new Jenkins job [...] 

Is Jenkins the right tool here? As a RM, I see no benefit. To me, the ideal would be that all RMs got access to some VM on Lizard fulfilling the above criteria, and that I write some shell script in the release process that simply is copy-pasted into a terminal to do the IUK building in said VM over SSH.

I'm surprised that you think adapt+copy'n'pasted manual build steps can be better than automated builds. I disagree and feel the need to argue in favour of automation. With an automated build system implemented e.g. as a Jenkins job:

- Better handling of failures: a Jenkins "failed" status is more obvious that a non-zero exit code that your shell may not warn you about (and even if it does, the RM may miss it when it's 2am and they're trying to finish a part of the release process before going to bed).
- We have to make the whole thing truly automatic modulo some well-defined parameters. One can't tell the same with shell script snippets from our release process doc, that often need adapting, which requires the RM to reason correctly and without mistakes. In practice it follows that:
  - RMs tend to make mistakes, especially when they either do the task too often and thus occasionally stop thinking (you) or when they do the task not often enough to fully misunderstand the instructions (bertagaz or myself). In this case it particularly matters because the exact same build must be done twice (once locally, once on lizard).
  - RMs tend to adapt/fix such instructions locally without improving the doc. We've seen cases where neither you nor I ever followed the release process doc to the letter (e.g. in terms of ordering) in the real world, and then when the occasional RM tries to follow the doc, guess what, we notice it was never tested and can't possibly work. Anything that leaves room for such creativity (let's be nice with ourselves :) tends to be create a gap between theory and practice. With a Jenkins job, we can be 100% certain that the build was done as designed and documented, that it works, and this increases the chances it'll work next time the occasional RM prepares a release.
- Build artifacts and logs are stored, tracked and published. This gives us an audit trail in case something goes wrong. That audit trail can be inspected by other Tails developers who can help fix the problem, compared to your terminal window. This also makes it easier to reproduce problems because we know what exact code was run when the problem happened.
- We get something consistent with how we build and publish the released ISO (see MATCHING_JENKINS_BUILD_ID=XXX in the release process doc).
- We're getting a little bit closer to CI. Adding manual adapt'n'copy'n'paste shell scripts does exactly the opposite.

More generally, most arguments in favour of automating builds & releases (i.e. CI) work here. I guess I don't need to tell you about them :)

I'm open to not block on this for the initial implementation of #15281 but I would be unhappy if it remained done manually for too long; we're too good at postponing stuff to the famous second iteration™ that never happens. So I'd like the manual solution you propose to be implemented in a way that
naturally leans towards automation: e.g. a program, living in jenkins-tools.git, with a clear interface, that explicitly gets any input it cannot guess as parameters, and that exits with sensible exit codes. Even without running it on Jenkins it'll already address some of the issues with the copy'n'paste approach that I listed above.

This way I can much more easily debug and find workarounds if there's problems.

As long as you have write access to the code that this Jenkins job would run and it's deployed when you push without requiring a sysadmin to do anything, I don't see a huge difference but I see what you mean: there's one more level of indirection between you (as the RM) and the code that runs. My counter argument is that the manual approach you're advocating for makes it harder for anyone else to "debug and find workarounds if there's problems".

#4 - 02/26/2018 08:56 PM - anonym
Wow, I honestly feel dumb and embarrassed for my comment above: as you seem to have caught on to, I have recently had a few episodes of "drowning in abstractions/indirection/layers/blah" while debugging fundamentally simply problems, which I think overwhelmed me, and caused me to react defensively. Thanks for nicely articulating some timely reminders of why things are the way they are for overall good reasons! ;)

intrigeri wrote:

I'm open to not block on this for the initial implementation of #15281 but I would be unhappy if it remained done manually for too long: we're too good at postponing stuff to the famous second iteration™ that never happens. So I'd like the manual solution you propose to be implemented in a way that naturally leans towards automation: e.g. a program, living in jenkins-tools.git, with a clear interface, that explicitly gets any input it cannot guess as parameters, and that exits with sensible exit codes. Even without running it on Jenkins it'll already address some of the issues with the copy'n'paste approach that I listed above.

Fully agreed!

This way I can much more easily debug and find workarounds if there's problems.

As long as you have write access to the code that this Jenkins job would run and it's deployed when you push without requiring a sysadmin to do anything, I don't see a huge difference but I see what you mean: there's one more level of indirection between you (as the RM) and the code that runs.

Yes, this is an actual concern that affects me. It's another thing like the tagged/time-based APT snapshot system -- I'm able to fix about half the issues I encounter, but for the tricky stuff I often end up urgently needing your help close to release time. That's pretty stressful, which there is enough of at that point in time any way. I think a good enough remedy is to have you "on-call" for dealing with such problems for a few releases (incl. RCs, but less urgently) when deploying this -- under what terms is that possible, if at all?
This way I can much more easily debug and find workarounds if there's problems.

As long as you have write access to the code that this Jenkins job would run and it's deployed when you push without requiring a sysadmin to do anything, I don't see a huge difference but I see what you mean: there's one more level of indirection between you (as the RM) and the code that runs.

Yes, this is an actual concern that affects me. It's another thing like the tagged/time-based APT snapshot system -- I'm able to fix about half the issues I encounter, but for the tricky stuff I often end up urgently needing your help close to release time.

I have no data I could check about such situations but my feeling is that in these tricky cases, the kind of help you need is about helping you understand fine details of how the system works in corner cases so either you can workaround/fix our stuff to avoid hitting corner cases, or I will make our code handle such corner cases better. I doubt that running the code locally vs. remotely would make a big difference: without that understanding of these fine details, even if you could run/debug the code locally, you would sometimes not be in a position to decide what's a suitable fix. I think it'll be just the same for generating IUKs unless you learn enough Modern Perl and dive deep enough into our incremental upgrades design+implementation to be fully autonomous in this area, which IMO has a rather bad cost/benefit for Tails. Anyway, I don't have data to back this feeling and I suspect you don't either, so let's leave it at that given:

That's pretty stressful, which there is enough of at that point in time any way.

This I totally understand and I want to take it into account!

I think a good enough remedy is to have you "on-call" for dealing with such problems for a few releases (incl. RCs, but less urgently) when deploying this -- under what terms is that possible, if at all?

I don't understand why this would be needed specifically for the Jenkins deployment: as long as the RM can fallback to running/debugging/fixing the script locally, even if the Jenkins job does not do what the RM needs we're good, no? Or were you asking even for the case when Jenkins is not involved and the RM runs the script locally?
Next step: specify the dependencies, input and output of the script. Leaving this on anonym's plate for now but I could take over this step if it's one task too many for you.

Once we have this we can:

- find someone to implement it (I'm thinking of our new FT colleagues)
- design the Jenkins job that will run this script (e.g. it might be that the script's input includes info that's too hard for a program to guess, and then the job will need whoever runs it to fill some parameters that'll be converted to input for the script)
This is not on our roadmap.

What matters is not particularly that this is done on Jenkins, it's that this is done on a machine from which lizard can quickly download a big pile of IUKs.

Input:

- commit of tails-iuk to checkout (otherwise it'll be too hard to test things like #17262, and to validate code changes with this new CI job before merging them); if possible, make this optional and default to master
- commit of tails-perl5lib to checkout (same as tails-iuk)
- SOURCE_DATE_EPOCH
- version of Tails the generated IUK shall upgrade to
- list of Tails versions the generated IUK shall upgrade from (with #15281 these will become "initially installed versions"; until then, they are "currently running version")
- We probably need the possibility to specify extra arguments that will be passed to tails-create-iuk. E.g. #9373 adds --union-type (aufs|overlayfs), and tails-create-iuk refuses running if we pass it args it does not support.
I have a working PoC: https://jenkins.tails.boum.org/job/build_IUKs/.

Known issue: workspace clean up fails, which breaks the next build on the same ISO builder. I think it’s because some temporary files are owned by root so the wrapper script should clean this up itself using sudo, or something.

#30 - 12/08/2019 09:19 AM - intrigeri
- Target version changed from Tails_4.3 to Tails_4.2

Technically, the first time we’ll really need this is during the 4.3 release process (assuming #15281 makes it into 4.2), but I’d really like to have something ready enough so I can test this during the 4.2 release process, so if anything goes wrong, I have time to fix things up.

#31 - 12/08/2019 09:30 AM - intrigeri

intrigeri wrote:

Known issue: workspace clean up fails, which breaks the next build on the same ISO builder. I think it’s because some temporary files are owned by root so the wrapper script should clean this up itself using sudo, or something.

This only happened when passing incorrect arguments to tails-create-iuk from the wrapper script, which was fixed then ⇒ case closed.

Next steps:
1. Design how to transfer the CI-built IUKs to rsync.lizard and validate them
2. Implement + document the above
3. Try this out during the 4.2 release process

#32 - 12/17/2019 12:02 PM - intrigeri
- Related to Bug #17361: Streamline our release process added

#33 - 12/18/2019 11:20 AM - intrigeri

FTR, https://jenkins.tails.boum.org/job/build_IUKs/31/ reproduced 2 IUKs that kibi built locally and published earlier this week, that upgrade systems to 4.1.1!

Here are the build parameters I’ve set for this Jenkins build:

```
IUK_COMMIT=3.5
PERL5LIB_COMMIT=Tails-perl5lib_2.0.2
SOURCE_DATE_EPOCH=1576450285
NEW_VERSION=4.1.1
SOURCE_VERSIONS=4.0 4.1
```

05/06/2020
#34 - 12/24/2019 12:33 PM - intrigeri
- Parent task deleted (#15281)

Let's allow ourselves to close #15281 even if this is not done in time for 4.2.

#35 - 12/24/2019 12:34 PM - intrigeri
- Related to Feature #15281: Stack one single SquashFS diff when upgrading added

#36 - 12/25/2019 11:41 AM - intrigeri
Updated next steps:

1. adjust wrap_tails_create_iuks to the fact the iuk & perl5lib code bases are moving to tails.git
2. design how to transfer the CI-built IUKs to rsync.lizard and validate them
3. implement + document the above
4. try this out during the 4.2 release process

#37 - 12/26/2019 11:24 AM - intrigeri
intrigeri wrote:
intrigeri wrote:

Known issue: workspace clean up fails, which breaks the next build on the same ISO builder. I think it's because some temporary files are owned by root so the wrapper script should clean this up itself using sudo, or something.

This only happened when passing incorrect arguments to tails-create-iuk from the wrapper script, which was fixed then ⇒ case closed.

Nope, this also happened on success. Fixed with 15016398d216a7f1512bab321064bcd5524db4.

#38 - 12/29/2019 08:04 PM - intrigeri
- Related to Feature #17385: Grow /home on rsync.lizard added

#39 - 12/29/2019 08:28 PM - intrigeri
- Status changed from In Progress to Needs Validation
- Feature Branch set to feature/15281-single-squashfs-diff

intrigeri wrote:

Updated next steps:
1. adjust wrap_tails_create_iuks to the fact the iuk & perl5lib code bases are moving to tails.git
2. design how to transfer the CI-built IUKs to rsync.lizard and validate them
3. implement + document the above

Done. Next steps:

1. The whole branch this is part of should be merged in time for 4.2 (so I expect segfault will do a quick review of what I did here). @anonym, if you have some time for this, I would love your opinion too on the commits that reference this ticket: you're a RM and we've designed the reproducibility verification stuff together initially :)
2. I'll try this out during the 4.2 release process and will fix the problems I notice then. It'll be a bit tricky because we will only generate IUK v1 for 4.2, but I'll manage, somehow. In any case, I've committed to be around to help @anonym when he goes through this process, during the 4.3 release process.

#40 - 01/06/2020 06:45 PM - intrigeri
- Status changed from Needs Validation to In Progress

Applied in changeset tails|d163abc00247672b75bb92449c57c91e27c51e03.

#41 - 01/06/2020 07:54 PM - intrigeri
- Status changed from In Progress to Resolved
- Assignee deleted (intrigeri)

I just had to adjust the doc a little bit; apart of that, it went well and I could publish Jenkins’ IUKs!

#42 - 01/09/2020 09:39 AM - intrigeri
- Related to Feature #17412: Drop the need for dedicated temporary storage space for IUKs on rsync.lizard added