Continuous Integration of The FreeBSD Project

Li-Wen Hsu
The FreeBSD Project
lwhsu@FreeBSD.org

Abstract

The FreeBSD project [22]'s continuous integration project started in the late 2013. We use Jenkins [17] automation server to build our continuous integration system. It monitors the svn repository for new commits and triggers a new build of it. In each build, the build machine compiles the latest code, creates disk image and creates a virtual machine to run test suite. In the meantime, we collect the compiler warnings and perform some further checks like clang analyzer. All these information are published to the developers and users to improve the quality of the FreeBSD project. In this paper, we describe the details of the system implementation.

1 Introduction

The term of continuous integration (CI) is first used by Grady Booch in 1991 [1], then Extreme programming (XP) extends the concept of CI and suggests to integrate as many times as possible per day. Now, CI is an important and common practice of software engineering. The benefits of CI are developers can know the conflicts and bugs earlier. This reduces the cost for resolving them later. Also, the status of the project is clearer and it is easier to estimate progress.

There are many software designed for various CI need, among them, the most popular one is Jenkins, an open source automation server written in Java, created by Kohsuke Kawaguchi. We use Jenkins as the main component, along with some other daemons like Pure-FTPd [20], to build FreeBSD project's CI system.

We describe the history of FreeBSD's CI work, how we trigger build from changes in version control system, do tests and publish results in the rest of this paper.

2 Automatic Build and History of CI work in FreeBSD

Dag-Erling Smrgrav created the first pubic CI system for the FreeBSD project: `http://tinderbox.FreeBSD.org`, this can be traced back in the mail archive in March 2002. It is believed the first continuous integration service of the FreeBSD project. Tinderbox is an open source project, codes are available at `svn://svn.freebsd.org/base/user/des/tinderbox`. Documents can be found in FreeBSD doc project [24] and wiki [25]. Unfortunately this service stopped in September or October 2014, nevertheless, here are many scripts from this project derived or inspiring the successive projects.

In late 2013, Craig Rodrigues founded jenkins-admin@FreeBSD.org with several developers. The main duty of jenkins-admin is maintaining the Jenkins instance in FreeBSD cluster, `https://jenkins.FreeBSD.org`. Which is an independent successor of tinderbox.FreeBSD.org. We configure it monitors the FreeBSD src and doc repositories, and triggers a new build when there is new commit in each supported branch. We created the first "build-and-test pipeline" for the FreeBSD project. In each build triggered by a change in repository, after compiling successes, we create a virtual machine disk image with the latest binaries, and spawn a new virtual machine to run tests. The test results are collected and stored by Jenkins server.

The highlight features of jenkins.FreeBSD.org are:

- Periodically build for head, stable-11, stable-10 and stable-9
- Architectures built against: amd64, i386, sparc64 and arm64
- Build FreeBSD with GCC, using amd64-xtoolchain-gcc
- Experimental Jenkins 2 "pipeline as code"

Shortly, jenkins.FreeBSD.org become a significant role in the FreeBSD development.

In 2016, with the hardwares sponsored by the FreeBSD foundation, we created `https://ci.FreeBSD.org`, an experimental site for testing new ideas. We still leverage Jenkins because it has proven useful on jenkins.FreeBSD.org. Aditionally, we implemented:

- Artifact server
- Job template

Having a centralized artifact server enables people retrieving the files generated from each stage of a pipeline, which can be used for reproducing issues for debugging purpose, or doing further tests. The details about job template are in "open configurations" section.

Another experiment in this system is we try to make the build pipeline more fine-grained, and introduced more build stages. For example, we not only create virtual machine disk image specialized for tests, but also try to produce the distribution files and virtual machine disk images the same as the official release provided by release engineering team . Also, we build LINT kernel for more test coverage.

There are multiple stages in a pipelne, each is a Jenkins job. The naming conversion of the build jobs is:

```
FreeBSD-{branch}-{target_arch}-{stage}
```

We create a pipeline for each supported branch and architecture. Take head branch and amd64 architecture for example:

- FreeBSD-head-amd64-build:
  Build the world and kernel from scratch and creates distribution files.
- FreeBSD-head-amd64-LINT:
  Build the LINT kernel
- FreeBSD-head-amd64-images:
  Build virtual machine disk images from distribution files with the same configuration of official release
- FreeBSD-head-amd64-testvm:
  Build virtual machine disk image from distribution files and test files and package are pre-installed.
- FreeBSD-head-amd64-test:
  Run the image from "testvm" stage in a new created virtual machine.

Their relationship is shown in 1

The codes are available at `https://github.com/lwhsu/freebsd-ci` and `https://github.com/lwhsu/jjb-freebsd-ci`, merging back to the main freebsd-ci repository on github or even `svn://svn.freebsd.org` is in progress.

3 Self Tests

In 2013 Rui Paulo created the `/usr/tests` hierarchy, with the Kyua [18] test framework by Julio Merino. These tests can be performed by command:

```
cd /usr/tests && kyua test
```

As of January 2017, we have 6023 tests for head branch on amd64 architecture.

As previous section mentioned, we build virtual machine disk images with the required files and packages for running tests. We pre-install the kyua package and put a script which runs kyua commands, and generates the test reports after all tests are executed. Finally it issues shutdown command. After the disk image ready, for amd64 and i386 architecture, we launch a bhyve [27] virtual machine, with an expect script for starting running tests in, and also for catching timeout in case.

After the testing virtual machine stopped, we extracted the test results from the disk image, put to the workspace where Jenkins slave daemon will collect them and send back to the Jenkins master for archiving and maintaining a trend record. The script to create test virtual machine image is `scripts/build/build-test_image.sh`, and the main script for executing tests in virtual machines and extracting results is `scripts/test/run-tests.sh`.

4 Access to Results

We configure Kyua to generate results as JUnit [4] XML format, which is natively supported by Jenkins and many other tools. Jenkins default plugins take outputs from the builds, make it a friendly accessible web interface at `https://<job-url>/<build-number>/testReport` (fig. 2, 3), and maintains a historical data (fig. 4). This helps us tracking the software quality trending.

The artifact server, artifact.ci.FreeBSD.org is newly introduced in ci.FreeBSD.org, its main responsibility is to share files between jobs in a pipeline. The artifact server is basically an FTP server powered by Pure-FTPd for serving files uploading. For downloading, we use a nginx [28] web server to provide read-only access. For security reason, we configure FTP server can only be accessed by FTP over TLS. As a result, we also added FTP over TLS feature in "Publish Over FTP Plugin" [?] of Jenkins.

In the end of each stage, Jenkins uploads created files to the artifact server, there is only one root directory "snapshots" on artifact.ci.FreeBSD.org for now. The directory layout of it is: `{branch}/{svn_revision}/{target}/{target_arch}`, for example, `head/r303226/amd64/amd64` . This layout

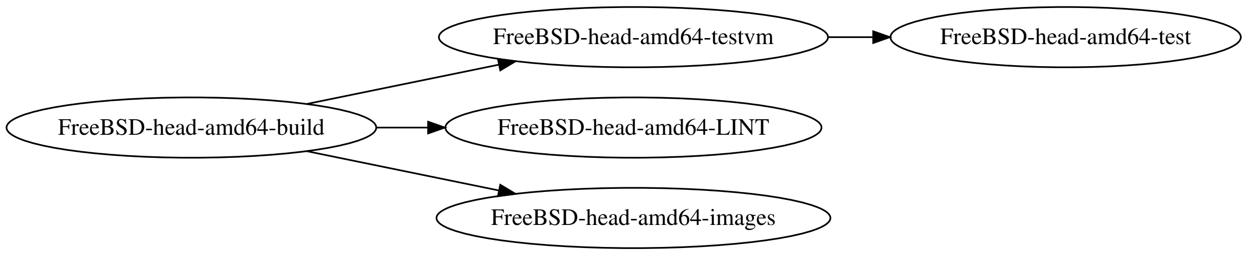
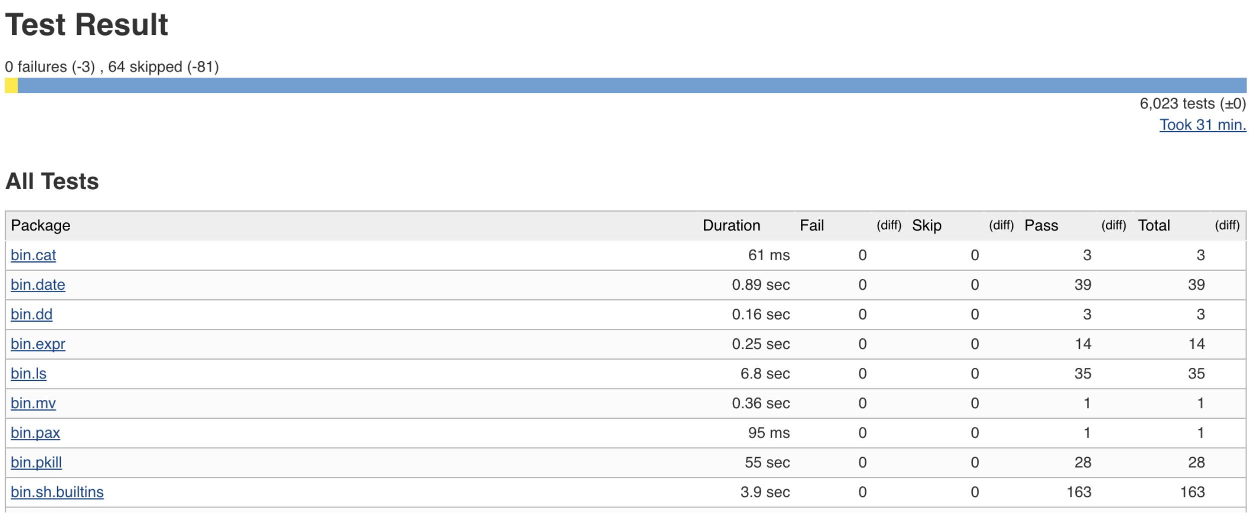


Figure 1: head-amd64 pipeline



Test Result

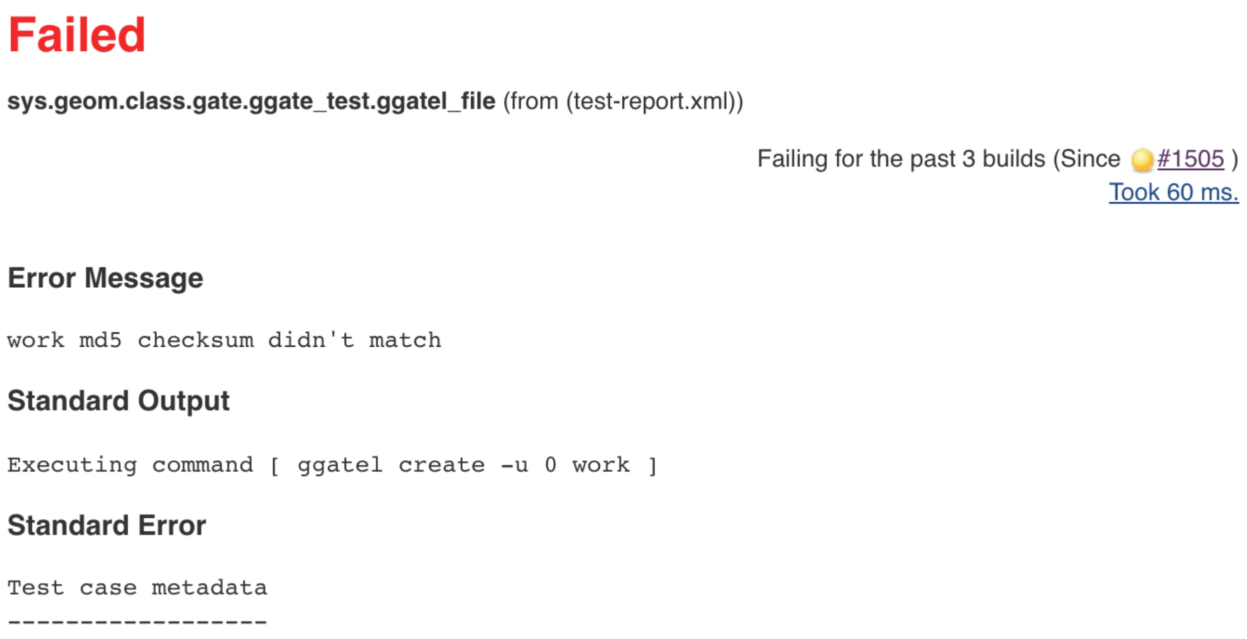
0 failures (-3) , 64 skipped (-81)

6,023 tests (±0)
Took 31 min.

All Tests

Package	Duration	Fail	(diff)	Skip	(diff)	Pass	(diff)	Total	(diff)
bin.cat	61 ms	0		0		3		3	
bin.date	0.89 sec	0		0		39		39	
bin.dd	0.16 sec	0		0		3		3	
bin.expr	0.25 sec	0		0		14		14	
bin.ls	6.8 sec	0		0		35		35	
bin.mv	0.36 sec	0		0		1		1	
bin.pax	95 ms	0		0		1		1	
bin.pkill	55 sec	0		0		28		28	
bin.sh.builtins	3.9 sec	0		0		163		163	

Figure 2: Test result from a test job



Failed

sys.geom.class.gate.ggate_test.ggatel_file (from (test-report.xml))

Failing for the past 3 builds (Since #1505)
Took 60 ms.

Error Message

```
work md5 checksum didn't match
```

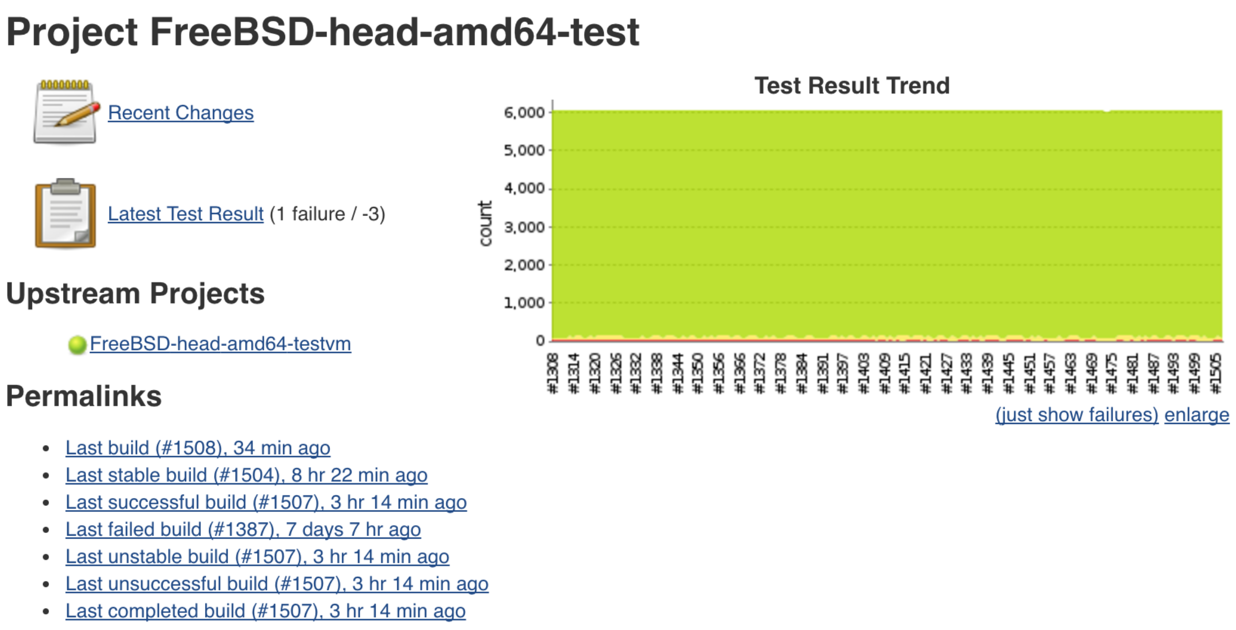
Standard Output

```
Executing command [ ggatel create -u 0 work ]
```

Standard Error

```
Test case metadata
------------------
```

Figure 3: Failed test case display


Project FreeBSD-head-amd64-test

Recent Changes

Latest Test Result (1 failure / -3)

Upstream Projects

FreeBSD-head-amd64-testvm

Permalinks

- Last build (#1508), 34 min ago
- Last stable build (#1504), 8 hr 22 min ago
- Last successful build (#1507), 3 hr 14 min ago
- Last failed build (#1387), 7 days 7 hr ago
- Last unstable build (#1507), 3 hr 14 min ago
- Last unsuccessful build (#1507), 3 hr 14 min ago
- Last completed build (#1507), 3 hr 14 min ago

Figure 4: Test result trend

is compatible with the official download site. We hope that in the future these snapshot artifacts can also be included and be more easily accessed by users.

Currently we have these artifacts for each pipeline:

- *.txz
  - From *-build jobs
  - Distribution files just like what on `https://download.FreeBSD.org`
- disk.img.xz
  - From *-images jobs
  - Virtual machine disk image file, installed from above *.txz files
- disk-test.img.xz
  - From *-testvm jobs
  - Virtual machine disk image file, with test script and test case

All files are publicly available at `http://artifact.ci.freebsd.org`.

5 Open Configurations

It is very important to make public can setup a highly identical instance of a public service, and security is also need to be considered. In the past, Jenkins job can only be edited on web interface, which is not easy to tracking the changes and reproduce. Users who has no administrative permission cannot see exactly configurations. Jenkins 2's "pipeline as code"' is a very promising to solve this problem. On jenkins.FreeBSD.org, we have tested it for the jobs which build head and stable-10 branch. The source code is available at `https://github.com/freebsd/freebsd-ci/blob/master/scripts/build/build-test.groovy`. However, currently not all the required Jenkins plugins of ci.FreeBSD.org supports this. We will look into this again and may switch it later.

We use an approach used by other project for a long time, Jenkins job builder [10], which converts YAML files to Jenkins XML configurations. And manipulate Jenkins instance via web service. With Jenkins job builder, the configuration of Jenkins is trackable and reproducible, and we can separate security credentials to other storage.

While working with other developers, we found the fact that not everyone knows (and has to know) how to configure Jenkins. So adding new build jobs requires Jenkins administrators configure and test build scripts by hand. This is a bottleneck of the development. We also found that most of the build jobs are just having different build scripts, other parts of the configurations such as repository URL and notification settings are the same. It inspired us to create a "job template" for similar jobs. A typical job contains following steps:

1. Environment setup
   (a) Check out latest source code

   (b) Setup needed version of FreeBSD

   (c) Install required packages

2. Execute specified build script for that job

3. Environment cleanup

The respective job definition (of Jenkins job builder) is very simple (listing 1):

Listing 1: Job definition

```
- job:
    name: FreeBSD-head-scan_build
    scm:
      - FreeBSD-src-head
    builders:
      - checkout-scripts
      - setup-jail
      - execute-in-jail
    publishers:
      - clean-jail
    wrappers:
      - timestamps
```

And we use jail to ensure the freshness of the build environment, and we can install "sudo" [21] program inside jail, with full privileges granted, for maximum the things user can do for testing. As a result, "build-in-jail script sets" is created, they are currently developed at `https://github.com/lwhsu/freebsd-ci/tree/jjb/scripts/jail`. It's first job is simplify environment setup and job execution, we have three scripts for each builder and publishers in definition of Jenkins job builder:

- setup.sh
  setup-jail builder, setup a jail jail and install require packages

- execute.sh
  execute-in-jail builder, execute build scripts from user

- clean.sh
  clean-jail publisher, clean the used jail to release resource

A typical job configuration contains three files:

- build.sh
  Main build script, which is executed by execue.sh in the jail

- jail.conf
  Jail environment configuration: version, arch, etc.

- pkg-list
  Package needs to be installed (from `pkg.FreeBSD.org`)

With these, adding a new job is much simplified. Submitter can just test their tested build script (build.sh), with some job definitions (jail.conf, pkg-list), and other configuration files, for example, src.conf or make.conf. Then jenkins-admin merges scripts to `freebsd-ci/jobs/<jobname>/`, and jenkins-admin creates a new job entry in jenkins job builder, push the new configuration to the Jenkins server. Everything is done. Our Jenkins server will execute it around the clock.

For example, FreeBSD-head-arm64-build contains files as listing 2, 3, 4:

Listing 2: Job configuration: jail.conf

```
TARGET=amd64
TARGET_ARCH=amd64
WITH_32BIT=0
OSRELEASE=10.3-RELEASE
```

Listing 3: Job configuration: pkg-list

```
aarch64-binutils
```

Listing 4: Job configuration: build.sh

```
#!/bin/sh
env \
  JFLAG=${BUILDER_JFLAG} \
  TARGET=arm64 \
  TARGET_ARCH=aarch64 \
  sh -x freebsd-ci/scripts/build/build-world-kernel.sh
```

We also add a "quarantine mode" of "build-in-jail script sets", just specify "QUARANTINE=1" in jail.conf, the the internet connection is removed after jail environment is setup, and resource is also strictly limited, such as execution time.

The other ci.FreeBSD.org setup can be found at `https://wiki.freebsd.org/Jenkins/Setup`.

6 Other Integrations

The best feature of Jenkins is its features can be extended by many plugins, we selected several plugins which considered useful for us and integrated into our system.

6.1 Notification

Make sure related developers can know the latest project status is very important. We use both Email and IRC notifications. The status for sending out notifications are:

- Build failed

- Build unstable (compile successfully, but some test cases failed)

- Build back to stable

And for notification email, we attach following information:

- Changes since last build (who & what)
- Tail of the console output
- What are the test cases failed
- Related URLs

Email notification is considered a "double-edged sword", once we send out too many mails and people will consider them as spam. If nobody reads the notification mails and takes action, it is totally no use. So use carefully created a pre-send script with "Email-ext plugin" [7], for filter out the internal errors of the jenkins cluster, to reduce false alert. The script is available at: `https://github.com/freebsd/freebsd-ci/blob/master/scripts/email-ext/pre-send.groovy`.

6.2 Code Review System

The FreeBSD project owns a code review system at `https://reviews.FreeBSD.org`, which is setup with Phabricator [19], and jenkins has a "Phabricator Differential Plugin" [11] plugin. It currently supports only git, we patched it to support subversion: `https://github.com/lwhsu/phabricator-jenkins-plugin/tree/scm`, and are also preparing to upstream it.

On ci.FreeBSD.org, we have two jobs:

- FreeBSD-head-amd64-build-phabric
- FreeBSD-doc-head-igor-phabric

They are using "quarantine mode" of the "build-in-jail script sets", and are triggered when a new patch is uploaded to reviews.FreeBSD.org (fig. 5).

And reports the build result back to `reviews.FreeBSD.org` (fig. 6).

6.3 Clang Static Analyzer

We created a "clang scan-build" job with Clang Static Analyzer [3] and the build script leveraged the "bsd.clang-analyzed.mk" from NetBSD [23], simply build the whole world and kernel with command in listing 5.

Listing 5: Command to perform clang scan-build

```
make analyze \
CLANG_ANALYZE_OUTPUT_DIR=clangScanBuildReports \
CLANG_ANALYZE_OUTPUT=html
```

The result is taken cared by "Clang Scan-Build Plugin" [6], which generates good web interface and history tracking (fig. 7, 8, 9)

6.4 Igor and Checkstyle

For document build, we use igor [26] by Warren Block, and Checkstyle [2] for checking the style of document XML. We worked with Warren and added a new option (-X) to igor, to output checkstyle XML. The respective job is FreeBSD-doc-head-igor. "Checkstyle Plugin" [5] of Jenkins also provides good web interface and history tracking. (fig 10, 11, 12)

6.5 Compiler Warnings

We also collect compiler warnings with "Warnings Plugin" [16]. For showing the trend of compiler warnings and this pushes developers produce higher quality code. (fig. 13, 14)

6.6 Build Status Badge

We provide build status badge for embedding in external web pages through "Embeddable Build Status Plugin" [8]. For example, on `https://wiki.FreeBSD.org/arm64` , we have badge as fig. 15

To show the latest status of each branch. The badge URLs are:

- `https://<job-url>/badge/icon`
- `https://<joburl>/lastCompletedBuild/badge/icon`

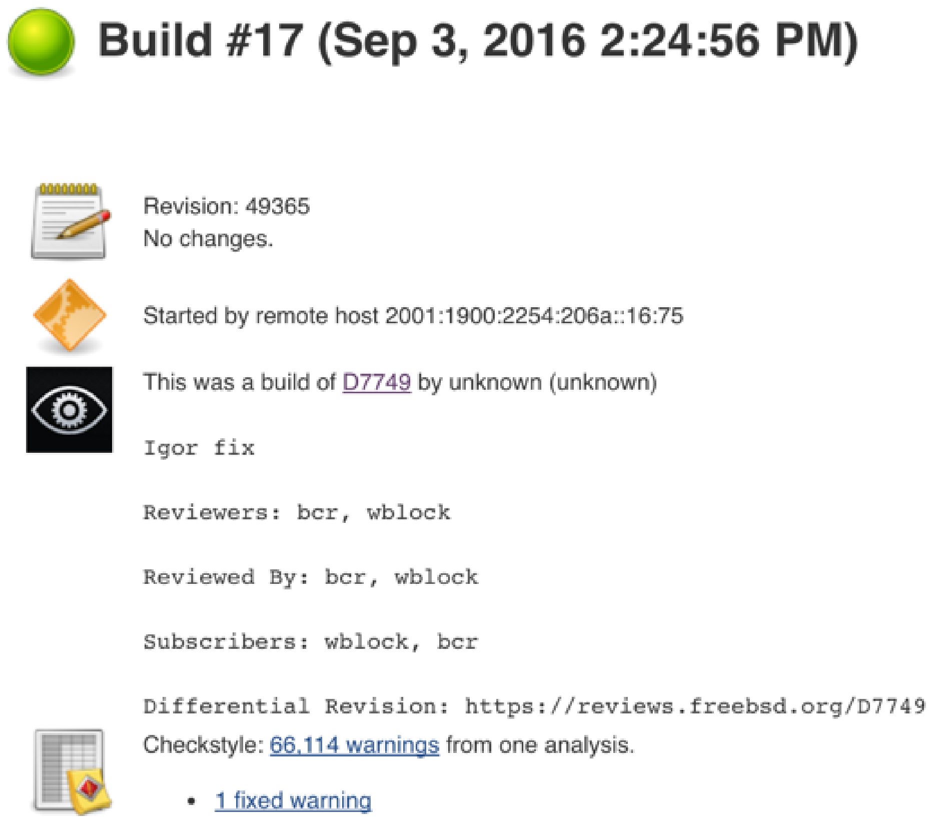
6.7 Others

We also use some other plugins for improving the interface and helping the maintenance task:

- SCM Sync configuration [14]
  - Use post-commit hook to notify administrators
- SafeRestart [15]
  - Schedule restart jenkins when all current jobs complete
  - Save and resume the build queue
- Green balls [9]
  - To replace blue status ball with green ones, to fulfill the convention for most culture.

7 Conclusion

In this paper, we described about the history and how we setup the latest CI system of the FreeBSD project. CI does help the development of the FreeBSD project, and we hope these experience and codes can also help downstream projects.


Build #17 (Sep 3, 2016 2:24:56 PM)

Revision: 49365
No changes.

Started by remote host 2001:1900:2254:206a::16:75

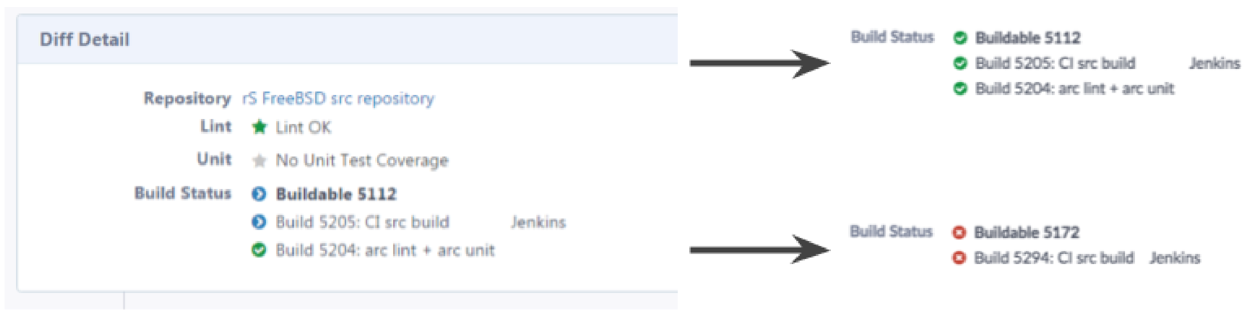
This was a build of D7749 by unknown (unknown)

```
Igor fix

Reviewers: bcr, wblock

Reviewed By: bcr, wblock

Subscribers: wblock, bcr

Differential Revision: https://reviews.freebsd.org/D7749
```

Checkstyle: 66,114 warnings from one analysis.

- 1 fixed warning

Figure 5: Build triggerd by a patch

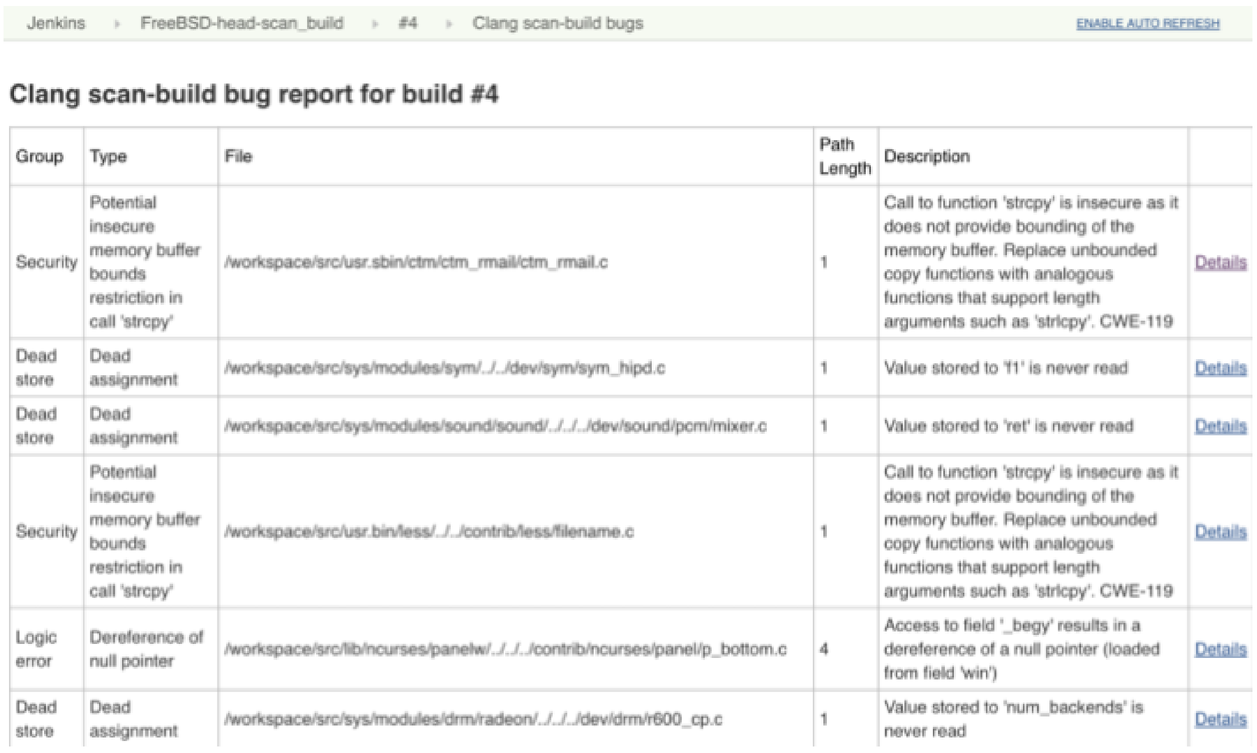

Diff Detail

Repository	rS FreeBSD src repository
Lint	Lint OK
Unit	No Unit Test Coverage
Build Status	Buildable 5112
	Build 5205: CI src build Jenkins
	Build 5204: arc lint + arc unit

Build Status	Buildable 5112
	Build 5205: CI src build Jenkins
	Build 5204: arc lint + arc unit

Build Status	Buildable 5172
	Build 5294: CI src build Jenkins

Figure 6: Rsult of a build triggerd by a patch



Jenkins › FreeBSD-head-scan_build › #4 › Clang scan-build bugs

ENABLE AUTO REFRESH

Clang scan-build bug report for build #4

Group	Type	File	Path Length	Description	
Security	Potential insecure memory buffer bounds restriction in call 'strcpy'	/workspace/src/usr.sbin/ctm/ctm_rmail/ctm_rmail.c	1	Call to function 'strcpy' is insecure as it does not provide bounding of the memory buffer. Replace unbounded copy functions with analogous functions that support length arguments such as 'strlcpy'. CWE-119	Details
Dead store	Dead assignment	/workspace/src/sys/modules/sym/../../dev/sym/sym_hipd.c	1	Value stored to 'f1' is never read	Details
Dead store	Dead assignment	/workspace/src/sys/modules/sound/sound/../../../dev/sound/pcm/mixer.c	1	Value stored to 'ret' is never read	Details
Security	Potential insecure memory buffer bounds restriction in call 'strcpy'	/workspace/src/usr.bin/less/../../contrib/less/filename.c	1	Call to function 'strcpy' is insecure as it does not provide bounding of the memory buffer. Replace unbounded copy functions with analogous functions that support length arguments such as 'strlcpy'. CWE-119	Details
Logic error	Dereference of null pointer	/workspace/src/lib/ncurses/panelw/../../../contrib/ncurses/panel/p_bottom.c	4	Access to field '_begy' results in a dereference of a null pointer (loaded from field 'win')	Details
Dead store	Dead assignment	/workspace/src/sys/modules/drm/radeon/../../../dev/drm/r600_cp.c	1	Value stored to 'num_backends' is never read	Details

Figure 7: Clang scan-build result

8 Future Work

The most important future work is: have more people to work on testing FreeBSD, providing more build and test scripts. And there are some other ideas:

- CI for ports
  Collaboration with "redports"
  Automatically "exp-run"
- Check for reproducible build
- More tests, even some are not enabled by default
  Dtrace
- Build for project branches
  Make testing feature branches easier
- Performance tests
- Work with other projects

We hope these work can make the quality of FreeBSD even better.

Acknowledgments

We want to thank following people, for sponsoring energy, idea and hardware:

- jenkins-admin@FreeBSD.org
- The FreeBSD Foundation
- clusteradmin@FreeBSD.org
- phabic-admin@FreeBSD.org
- People on -testing, -current, -stable mailing lists

References

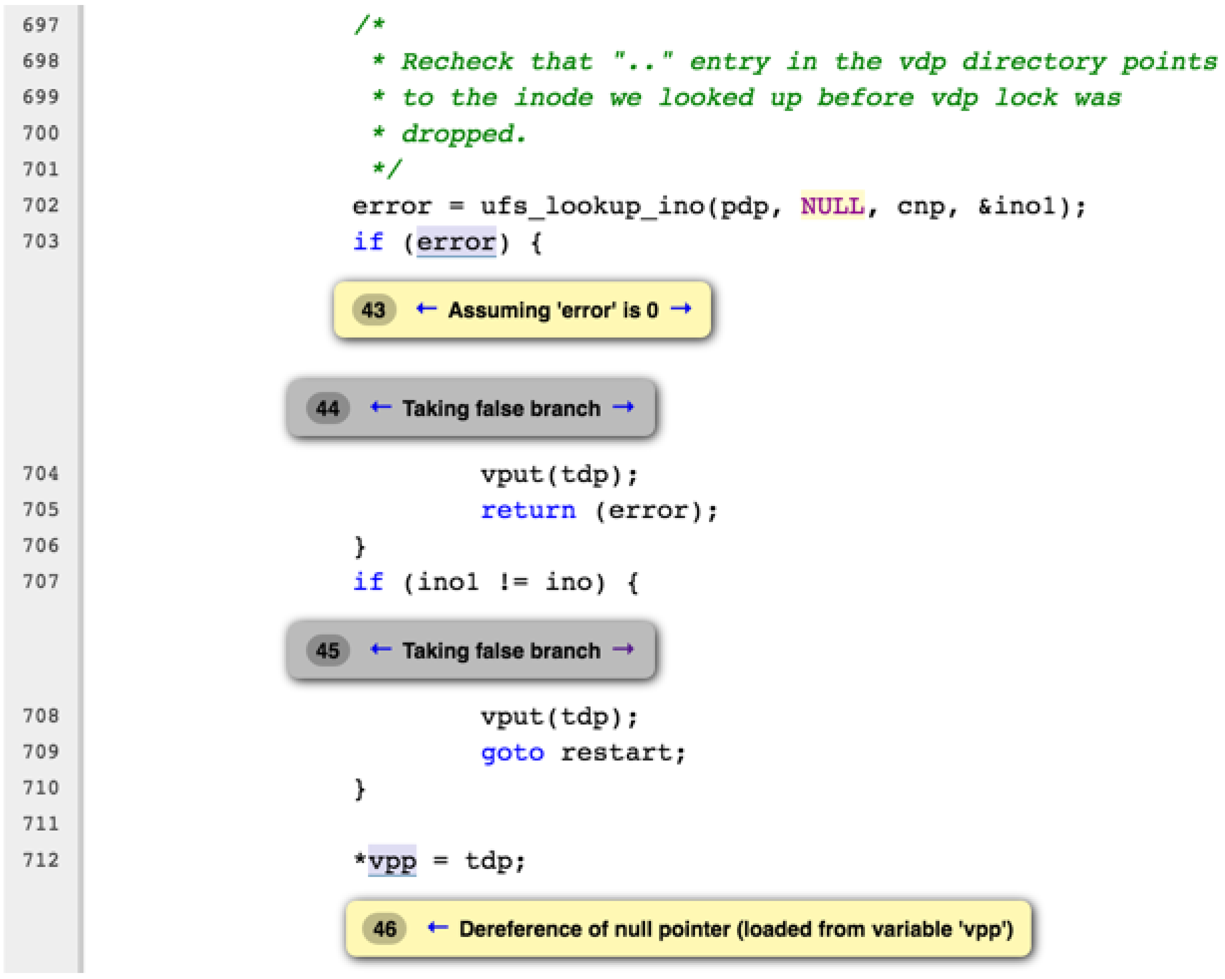
[1] Benjamin Cummings, Booch, Grady, *Object Oriented Design: With Applications*, 1991

[2] Checkstyle, `http://checkstyle.sourceforge.net`

[3] Clang Static Analyzer, `https://clang-analyzer.llvm.org`

[4] JUnit, `http://junit.org`

[5] Jenkins Checkstyle Plugin, `https://wiki.jenkins-ci.org/display/JENKINS/Checkstyle+Plugin`



```
        /*
         * Recheck that ".." entry in the vdp directory points
         * to the inode we looked up before vdp lock was
         * dropped.
         */
        error = ufs_lookup_ino(pdp, NULL, cnp, &ino1);
        if (error) {
    43 ← Assuming 'error' is 0 →
   44 ← Taking false branch →
                vput(tdp);
                return (error);
        }
        if (ino1 != ino) {
   45 ← Taking false branch →
                vput(tdp);
                goto restart;
        }

        *vpp = tdp;
    46 ← Dereference of null pointer (loaded from variable 'vpp')
```

Figure 8: Clang scan-build result explanation

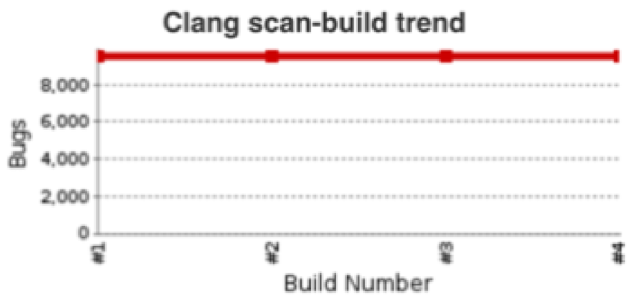
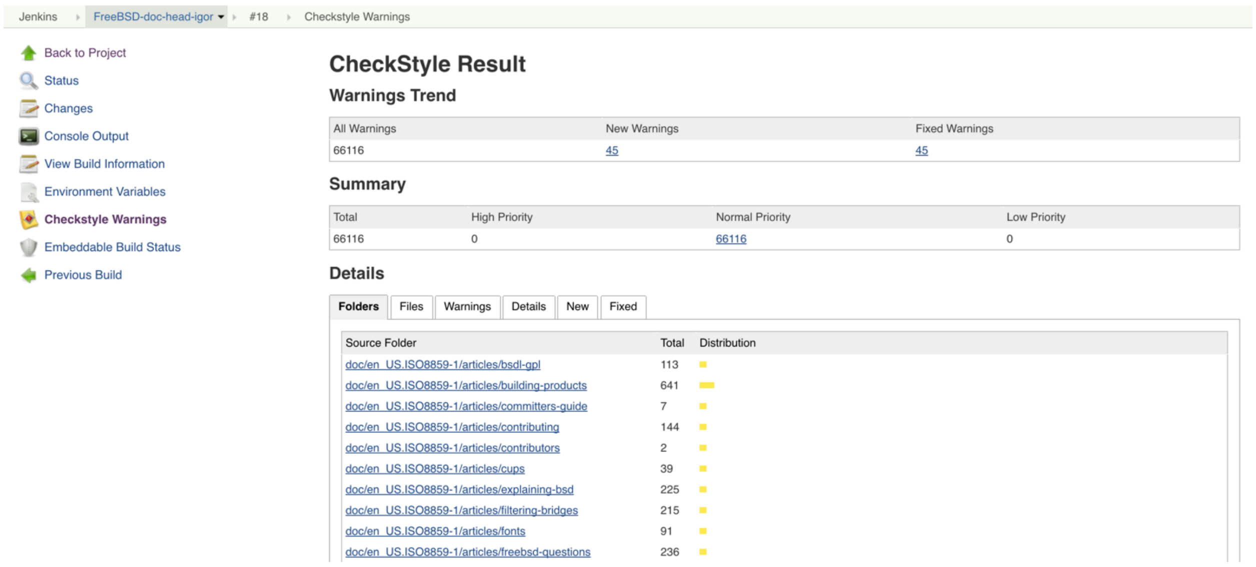


Figure 9: Clang scan-build trend



Jenkins › FreeBSD-doc-head-igor › #18 › Checkstyle Warnings

- Back to Project
- Status
- Changes
- Console Output
- View Build Information
- Environment Variables
- **Checkstyle Warnings**
- Embeddable Build Status
- Previous Build

CheckStyle Result

Warnings Trend

All Warnings	New Warnings	Fixed Warnings
66116	45	45

Summary

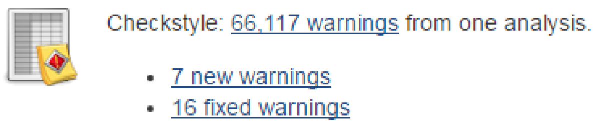
Total	High Priority	Normal Priority	Low Priority
66116	0	66116	0

Details

Folders | Files | Warnings | Details | New | Fixed

Source Folder	Total	Distribution
doc/en_US.ISO8859-1/articles/bsdl-gpl	113	🟨
doc/en_US.ISO8859-1/articles/building-products	641	🟨
doc/en_US.ISO8859-1/articles/committers-guide	7	🟨
doc/en_US.ISO8859-1/articles/contributing	144	🟨
doc/en_US.ISO8859-1/articles/contributors	2	🟨
doc/en_US.ISO8859-1/articles/cups	39	🟨
doc/en_US.ISO8859-1/articles/explaining-bsd	225	🟨
doc/en_US.ISO8859-1/articles/filtering-bridges	215	🟨
doc/en_US.ISO8859-1/articles/fonts	91	🟨
doc/en_US.ISO8859-1/articles/freebsd-questions	236	🟨

Figure 10: Igor + Checksytle - result



Checkstyle: 66,117 warnings from one analysis.

- 7 new warnings
- 16 fixed warnings

Figure 11: Igor + Checkstyle - comparing with previous build

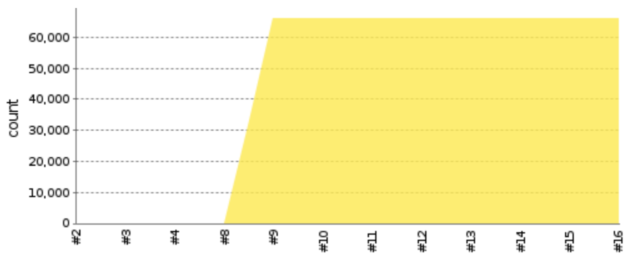
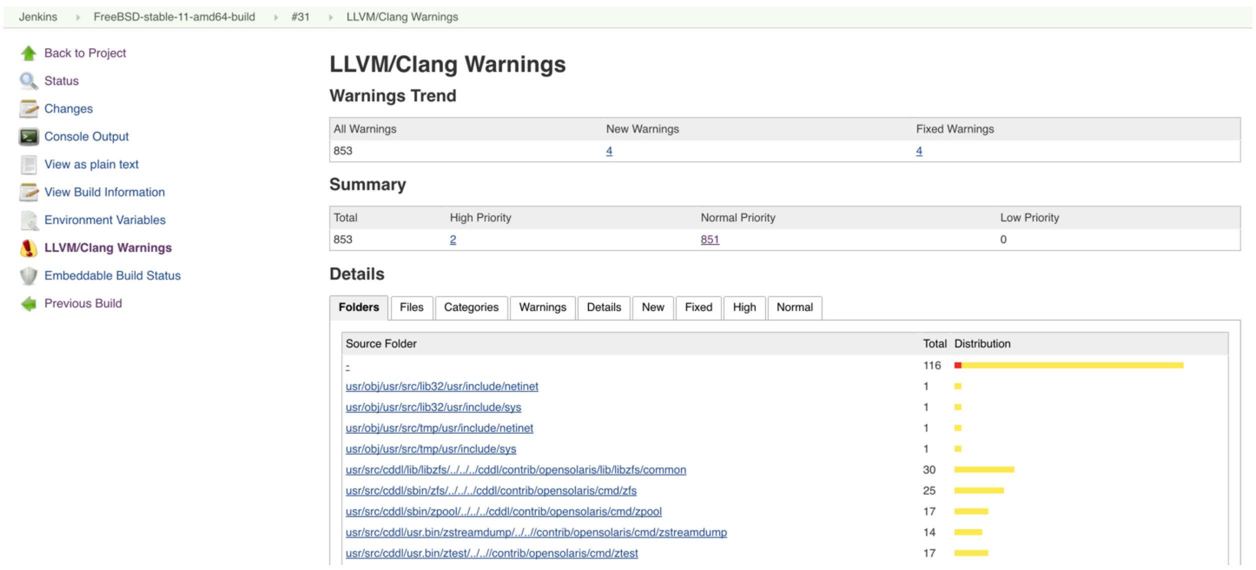


Figure 12: Igor + Checkstyle - trend



Jenkins ▸ FreeBSD-stable-11-amd64-build ▸ #31 ▸ LLVM/Clang Warnings

- Back to Project
- Status
- Changes
- Console Output
- View as plain text
- View Build Information
- Environment Variables
- **LLVM/Clang Warnings**
- Embeddable Build Status
- Previous Build

LLVM/Clang Warnings

Warnings Trend

All Warnings	New Warnings	Fixed Warnings
853	4	4

Summary

Total	High Priority	Normal Priority	Low Priority
853	2	851	0

Details

Folders | Files | Categories | Warnings | Details | New | Fixed | High | Normal

Source Folder	Total	Distribution
-	116	
usr/obj/usr/src/lib32/usr/include/netinet	1	
usr/obj/usr/src/lib32/usr/include/sys	1	
usr/obj/usr/src/tmp/usr/include/netinet	1	
usr/obj/usr/src/tmp/usr/include/sys	1	
usr/src/cddl/lib/libzfs/../../../cddl/contrib/opensolaris/lib/libzfs/common	30	
usr/src/cddl/sbin/zfs/../../../cddl/contrib/opensolaris/cmd/zfs	25	
usr/src/cddl/sbin/zpool/../../../cddl/contrib/opensolaris/cmd/zpool	17	
usr/src/cddl/usr.bin/zstreamdump/../..//contrib/opensolaris/cmd/zstreamdump	14	
usr/src/cddl/usr.bin/ztest/../..//contrib/opensolaris/cmd/ztest	17	

Figure 13: Compiler Warnings - result

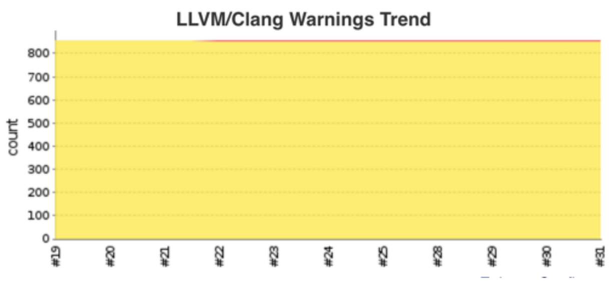


Figure 14: Compiler Warnings - trend

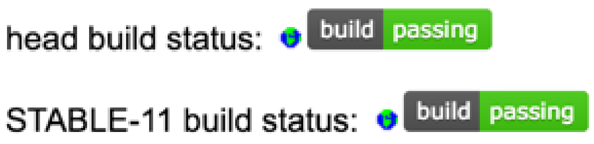


Figure 15: Build satus badge

[6] Jenkins Clang Scan-Build Plugin,
`https://wiki.jenkins-ci.org/display/JENKINS/Clang+Scan-Build+Plugin`

[7] Jenkins Email-ext plugin,
`https://wiki.jenkins-ci.org/display/JENKINS/Email-ext+plugin`

[8] Jenkins Embeddable Build Status Plugin,
`https://wiki.jenkins-ci.org/display/JENKINS/Embeddable+Build+Status+Plugin`

[9] Jenkins Green Balls Plugin,
`https://wiki.jenkins-ci.org/display/JENKINS/Green+Balls`

[10] Jenkins Job Builder,
`http://docs.openstack.org/infra/jenkins-job-builder/`

[11] Jenkins Phabricator Differential Plugin,
`https://wiki.jenkins-ci.org/display/JENKINS/Phabricator+Differential+Plugin`

[12] Jenkins Publish Over FTP Plugin,
`https://wiki.jenkins-ci.org/display/JENKINS/Publish+Over+FTP+Plugin`

[13] Jenkins SCM Sync configuration plugin,
`https://wiki.jenkins-ci.org/display/JENKINS/SCM+Sync+configuration+plugin`

[14] Jenkins SCM Sync configuration plugin,
`https://wiki.jenkins-ci.org/display/JENKINS/SCM+Sync+configuration+plugin`

[15] Jenkins SafeRestart Plugin,
`https://wiki.jenkins-ci.org/display/JENKINS/SafeRestart+Plugin`

[16] Jenkins Warnings Plugin,
`https://wiki.jenkins-ci.org/display/JENKINS/Warnings+Plugin`

[17] Jenkins,
`https://jenkins.io`

[18] Merino, Julio, Kyua, Testing framework for infrastructure software,
`https://github.com/jmmv/kyua`

[19] Phabricator,
`https://www.phacility.com/phabricator/`

[20] Pure-FTPd,
`https://www.pureftpd.org`

[21] Sudo,
`http://www.sudo.ws`

[22] The FreeBSD Project,
`https://www.FreeBSD.org`

[23] The NetBSD Project,
`http://www.netbsd.org`

[24] Tinderbox, FreeBSD Developers' Handbook,
`https://www.freebsd.org/doc/en/books/developers-handbook/testing-tinderbox.html`

[25] Tinderbox, FreeBSD Wiki,
`https://wiki.freebsd.org/Tinderbox`

[26] Warren Block, *igor, FreeBSD Documentation Project sanity check script*,
`http://www.wonkity.com/~wblock/igor/`

[27] bhyve(8), FreeBSD System Manager's Manual,
`https://man.freebsd.org/bhyve`

[28] nginx,
`https://nginx.org/`