# GIT Commands

Saturday, June 24, 2017 3:57 AM

To add files to git:
```
    git add file.txt
```

To remove files:
```
    git rm '*.txt'
```

To pull remote changes to local repository:
```
    git pull origin
```

In case of merge conflict:

```
    git mergetool
```

To show changes since last commit:
```
    git diff HEAD
```

To show difference between staged changes and the HEAD of repo:
```
    git diff --staged
```

To cancel staged changes (cancel changes):
```
    git reset octofamily/octodog.txt
```

Note: The file changes are still there on disk (the changes are not reverted).

To revert file back to committed content (undo changes of file):
```
    git checkout -- octocat.txt
```

To create branch:
```
    git branch my_branch
```

Note: Usually branch will be merged back to master branch later.

To create a new branch and switch to it at the same time, you can run the git checkout command with the -b switch:

```
$ git checkout -b iss53
Switched to a new branch "iss53"
```

This is shorthand for:
```
$ git branch iss53
$ git checkout iss53
```

To create branch and switch to it in one command:
```
    git checkout -b my_branch
```

To switch to branch:
```
    git checkout my_branch
```

To commit branch changes (same as master branch/trunk):
```
    git commit -m 'My changes'
```

To switch back to master branch:
```
    git checkout master
```

To merge changes from branch into master branch:

*(Need to switch to master branch first, as above.)*
```
    git merge my_branch
```

To delete local branch:
```
    git branch -d my_branch
```

To delete remote branch:
```
    git push origin --delete <branch_name>
```

To push current branch's changes back to remote repo:
```
    git push -u origin
```

To push specific branch's changes back to remote repo:
```
    git push -u origin v1.2
```

To create tag (1.2.0):

```
    git tag 1.2.0
```

You will need to push tag to server explicitly:

```
    git push origin <tagname>
```

If you have a lot of **tags** that you want to **push** up at once, you can also use the --**tags** option to the **git push** command.

To view commit logs in nice format
```
    git log --pretty --oneline --graph
```

**To delete remote branch and local branch:**
```
git push origin --delete <branch_name> - For remote branch
git branch -d <branch_name> - for local branch
```

**To list all branches:**
**git branch -r** - For remote
**git branch -a** - For local
NOTE: Branches displayed in RED color means those branches are not pulled locally.

Make an existing Git branch track a remote branch?
Why do I need to do `--set-upstream` all the time?
**git branch -u origin/your_branch_name - Use this command to track local branch to the remote branch. Please note that remote branch should already exist.**
**git push --set-upstream origin my_branch<ex:version199>**

**To abort merge process:** git reset --hard HEAD

**Git tip: How to "merge" specific files from another branch**
Git checkout <branch_name_to_which_changes_has_to_be_merged>
Git chekout <branch_name_from_which_changes_has_to_be_merged> <path_to_file_name_which_has_changes>

Hard reset local branch with remote changes: **git reset --hard origin/<branch_name>**

**When you get the following error:**
fatal: refusing to merge unrelated histories (You'll see this error when you merge changes to master branch after a very long time)

In order to overwrite all the changes in master branch with the changes in your new branch please use the command as shown below:
**git merge -X theirs --allow-unrelated-histories <branch_name_which_you_want_to_merge>**

**Exclude few files while merging**
**First you have to set a global command on your laptop:** git config --global merge.ours.driver true
Follow this link to get complete understanding.

Let's say you want to exclude the file `config.php`
On branch A:

1. Create a file named '.gitattributes' in the same dir, with this line: config.php merge=ours. This tells git what strategy to use when mergin the file. In this case it always keep your version, ie. the version on the branch you are merging into.
2. Add the .gitattributes file and commit

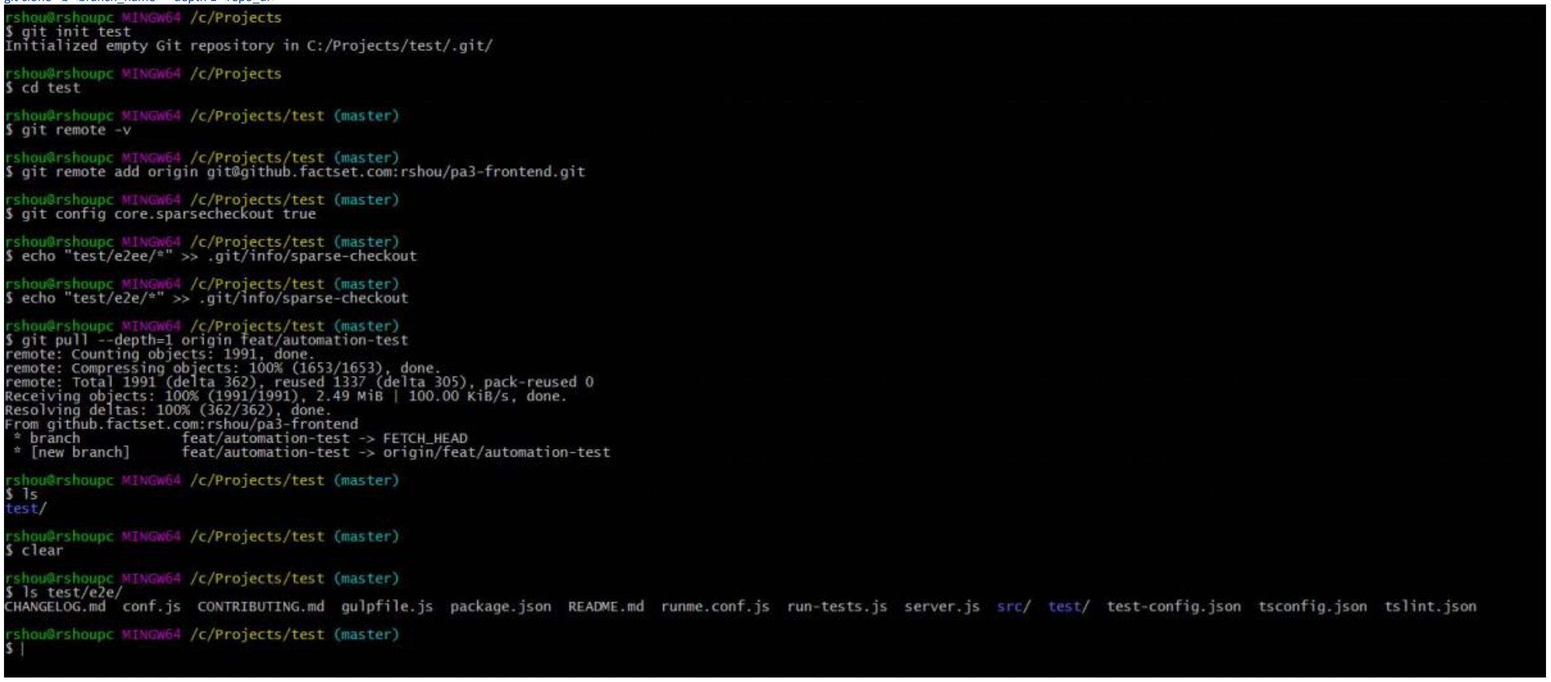
On branch B: repeat steps 1-2
Try merging now. Your file should be left untouched.

**Command to go back to previous branch your checked out:** git checkout -

**How to make local branch track the remote branch?**
git branch --set-upstream-to origin/<remote_branch_name_that_your_local_branch_should_track>
git branch --track branch-name origin/branch-name

**git checkout .** - Removes all modified files tracked files of your current branch
https://stackoverflow.com/questions/22620393/various-ways-to-remove-local-git-changes

**Cloning specific branch:**
git clone –b <branch_name> --depth 1 <repo_ur>



```
rshou@rshoupc MINGW64 /c/Projects
$ git init test
Initialized empty Git repository in C:/Projects/test/.git/

rshou@rshoupc MINGW64 /c/Projects
$ cd test

rshou@rshoupc MINGW64 /c/Projects/test (master)
$ git remote -v

rshou@rshoupc MINGW64 /c/Projects/test (master)
$ git remote add origin git@github.factset.com:rshou/pa3-frontend.git

rshou@rshoupc MINGW64 /c/Projects/test (master)
$ git config core.sparsecheckout true

rshou@rshoupc MINGW64 /c/Projects/test (master)
$ echo "test/e2ee/*" >> .git/info/sparse-checkout

rshou@rshoupc MINGW64 /c/Projects/test (master)
$ echo "test/e2e/*" >> .git/info/sparse-checkout

rshou@rshoupc MINGW64 /c/Projects/test (master)
$ git pull --depth=1 origin feat/automation-test
remote: Counting objects: 1991, done.
remote: Compressing objects: 100% (1653/1653), done.
remote: Total 1991 (delta 362), reused 1337 (delta 305), pack-reused 0
Receiving objects: 100% (1991/1991), 2.49 MiB | 100.00 KiB/s, done.
Resolving deltas: 100% (362/362), done.
From github.factset.com:rshou/pa3-frontend
 * branch            feat/automation-test -> FETCH_HEAD
 * [new branch]      feat/automation-test -> origin/feat/automation-test

rshou@rshoupc MINGW64 /c/Projects/test (master)
$ ls
test/

rshou@rshoupc MINGW64 /c/Projects/test (master)
$ clear

rshou@rshoupc MINGW64 /c/Projects/test (master)
$ ls test/e2e/
CHANGELOG.md  conf.js  CONTRIBUTING.md  gulpfile.js  package.json  README.md  runme.conf.js  run-tests.js  server.js  src/  test/  test-config.json  tsconfig.json  tslint.json

rshou@rshoupc MINGW64 /c/Projects/test (master)
$ |
```

Also if you want to checkout a specific directory from git repo, then you can take a look at sparseCheckout.
https://stackoverflow.com/questions/600079/how-do-i-clone-a-subdirectory-only-of-a-git-repository

which should allow you to download only the QA directories instead of whole repo.

**How to checkout only specific branch:**

1. First clone your repository as follows:
   **git clone -n <repo_url> <repo_name(example: qa-test-pa3)>**
2. Switch to <repo_name>
   **cd <repo_name>**
   **git config core.sparsechecout true** [To set this command globally execute: git config --global core.sparsechecout true]
   **echo * >> .git/info/sparse-checkout** [NOTE: "*" here will get all the files in current directory. If you want only specific folder from current directory then it should be **echo <path_to_directory/folder> >> .git/info/sparse-checkout**]
3. Now checkout the branch you want the version from
   Ex: **git checkout <branch_name>**

   **NOTE: All these commands should be executed in GIT Bash only.**
   **Source:** https://stackoverflow.com/questions/23289006/on-windows-git-error-sparse-checkout-leaves-no-entry-on-the-working-directory

**Second approach to clone only specific folder**

| | | |
|---|---|---|
| 1 | Git init | Initialize git |
| 2 | Git remote add origin <repo_url> | Add repository URL |
| 3 | Git config core.sparsecheckout true | Enable sparse-checkout |
| 4 | Echo * >> .git/info/sparse -checkout | Add folders to be pulled when **git pull** command is executed. In this case I have added "*" to pull all folders and files at root |
| 5 | Git pull --depth=1 origin | This command will pull all the folders and files from given branch |

## Shortcut command
**git clone -b <branch_name> --single-branch <repo_url>**

# git create/push tag

Monday, March 19, 2018 10:58 AM

```
git tag tagname
```

You will need to push tag to server explicitly:

```
git push origin <tagname>
```

If you have a lot of **tags** that you want to **push** up at once, you can also use the --**tags** option to the **git push** command.

# QA Test Development Git Workflow for JavaScript/Protractor

Wednesday, October 19, 2016 7:30 AM

## Git repo organization

Engineering PA3 repo: https://gitlab.factset.com/pa3/pa3-frontend
QA tests will be a subdirectory in engineering repo.

The current protractor and PA3 Gitlab repos are:
https://gitlab.factset.com/app-qa-automation/qa-protractor-common
https://gitlab.factset.com/app-qa-automation/qa-test-pa3

The PA3 test scripts are in its own Git repo, as PA3 has its own versioning, which is similar but not exactly the same as Online version. Having the test scripts in its own repo could help track PA3 versions better.

The PA3 Page Objects are included in the PA3 test script repo, since they tend to match the same product version.

The common files are in separate Git repo, because it is also used by other products, which are in separate Gitlab repos.

The repo and npm package naming are designed to be as short and clear as possible. And because they are company-wide names, we try to avoid naming collisions with other groups (by using prefix 'qa-test').

## Pre-requisite @fds setup

FDS registry setup

## Creating Project using Standard Template

Use qa-test-pru project as template.

## Building code

- Overall workflow, in typical order:

  ```
  gulp lint
  gulp build
  gulp test
  gulp docs
  git commit …
  gulp release
  ```

- You can also run `gulp` (without arguments), which will watch for file changes and automatically rebuild
- Project build system is company standard (@fds/gulp-node-lib-build), which is based on gulp, and it defines standard tasks, including build, lint, test, docs, watch, and release.
- Project source files are in src/ folder (standard structure)
- To build code:
  ```
  gulp build
  ```
  That will create built files in lib/.
  Note: It uses BabelJS to compile JavaScript, so it would support ES2015 (ES6).

- You can also let it watch for file changes and automatically build and lint:
  ```
  gulp
  ```

- To run lint check (syntax and style standards) separately:
  ```
  gulp lint
  ```

- git commit: Commit messages should follow company standards:
  http://thief.factset.com/idiomatic-git/commit-standards.html

- Release it:

  ```
  gulp release
  ```

  Then follow the prompts to set new version and update CHANGELOG.md when prompted.

  That will build, run unit test, commit to repo, tag new release, push changes to Gitlab server, using the current branch. It gets the repo ready for Jenkins, **but does not publish npm package.**

  **IMPORTANT:** `gulp release` should be your last commit (it will push the commit to server at the end). You must be in the right branch (git checkout) when you do that. Otherwise, Jenkins will not build your package.

## Lint Rules

As we are using the standard Developer Services build, the lint configuration comes from https://gitlab.factset.com/developer-services/lint-standards . It implements https://gitlab.factset.com/web-guidance/web-guidance/blob/master/javascript/quality-standard.md and https://gitlab.factset.com/web-guidance/web-guidance/blob/master/styleguides/javascript.md . If you have any questions about @fds/lint-standards, please file a Developer Services – Web Development Tools RPD.

## Rolling Branch Workflow

Rolling branch model diagram:

Git
Branching...

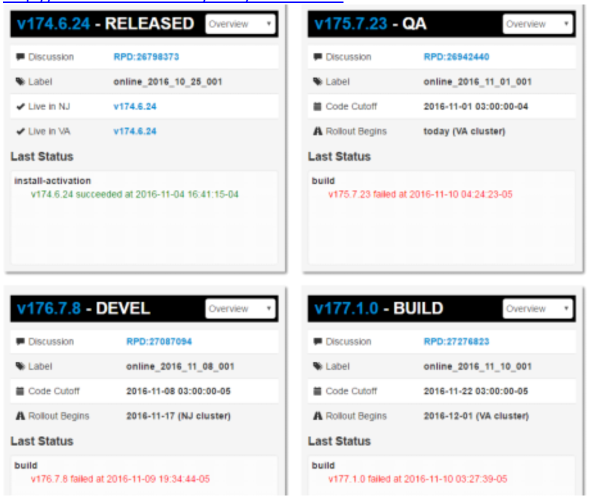
Git
Branching...

PA3 versions only matches the major version number of Online Tracker (for example, `v175`). That is, the minor version number of PA3 might differ from that in Online Tracker. PA3 source code repo might also have minor version branch like `v175.7` for features, but QA does not need to use the same minor version number.

We could use a different minor version number for each Stage:

| Stage | Minor version number |
|---|---|
| Prebuild/Edge Regression | 0 |
| Devel | 1 |
| QA | 2 |
| Live/Release | 3 |

**Online Tracker Example:**
http://tracker.factset.com/view/build-status



**v174.6.24 - RELEASED**

| | |
|---|---|
| Discussion | RPD:26798373 |
| Label | online_2016_10_25_001 |
| Live in NJ | v174.6.24 |
| Live in VA | v174.6.24 |

Last Status
install-activation
v174.6.24 succeeded at 2016-11-04 16:41:15-04

**v175.7.23 - QA**

| | |
|---|---|
| Discussion | RPD:26942440 |
| Label | online_2016_11_01_001 |
| Code Cutoff | 2016-11-01 03:00:00-04 |
| Rollout Begins | today (VA cluster) |

Last Status
build
v175.7.23 failed at 2016-11-10 04:24:23-05

**v176.7.8 - DEVEL**

| | |
|---|---|
| Discussion | RPD:27087094 |
| Label | online_2016_11_08_001 |
| Code Cutoff | 2016-11-08 03:00:00-05 |
| Rollout Begins | 2016-11-17 (NJ cluster) |

Last Status
build
v176.7.8 failed at 2016-11-09 19:34:44-05

**v177.1.0 - BUILD**

| | |
|---|---|
| Discussion | RPD:27276823 |
| Label | online_2016_11_10_001 |
| Code Cutoff | 2016-11-22 03:00:00-05 |
| Rollout Begins | 2016-12-01 (VA cluster) |

Last Status
build
v177.1.0 failed at 2016-11-10 03:27:39-05

We would have a version branch for each major version, for example 176 above.

## Merging back to master branch

After Live production, and it is working fine, merge the changes back into master branch:

```
git checkout master
git merge --no-ff Live

(and edit and fix any merge conflict)
(then commit again)
```

Note: It might be possible to use merge option to favor `their` changes, in the case of conflict, which would reduce chance of manual merge typos.

Run regression test to check for merge errors:

```
gulp lint
gulp build
gulp test
gulp docs
npm run regression
```

## Release

To release a new version, execute:

```
gulp release
```

It will prompt for new version and editing CHANGELOG.
It will then commit, create a tag named as the new version, and push the changes to git server.
It will trigger Jenkins build (and publish npm package to Artifactory) when you do `gulp release`, via Git Web Hook.

## Building and Publishing NPM Package

Manually building is usually not needed, but if you need to, this is how:

- Go to https://jenkins.factset.com/job/app-qa-automation/
- Select project qa-test-pru
- Click **Build Now**

That will build and publish npm package to Artifactory (http://artifactory.factset.com/).

## Code Convention

Most of the code convention comes from in company-standard JSHint configuration that is automatically run when you build in Jenkins.

- File encoding: UTF-8
- Use LF (Unix style) line separators, else lint will fail
- Configure git to disable converting LF line ending to CRLF:
  `git config --global core.autocrlf false`

  To see configuration:
  `git config -l --global`

- Indentation: 2 spaces (no tab characters)
- To quiet lint in protractor test scripts:
  ```
  var protractor = protractor || null;
  var browser = browser || null;
  ```
- Must have empty line after block (`{}`, `({})`, etc.).
  Example:
  ```
  if (…) {
  }

  // ^ Must have empty line above
  ```
- Line comments must be preceded with a blank line
- Disallow multiple var declaration, for example:
  `var err, stdinp;`
- Use single quotes around strings (double-quotes will get error)
- Statement must terminate with semicolon
- It is disallowed to declare variable but leave it unused
- No space after opening parenthesis and before closing parenthesis ('`( )`')

  ```
  browser.getAllWindowHandles().then(function(handles) {
  }, function(err) {
  });
  ```

  needs to be written with empty line above the second anonymous function:

  ```
  browser.getAllWindowHandles().then(function(handles) {
  },

  function(err) {
  });
  ```
- Operators (such as +) should be surrounded by space
- High cyclomatic complexity is disallowed:
  - Nested callback blocks have high cyclomatic complexity:

  ```
  fd.readdir(dir, function() {
    fs.readfile(file, encoding, function() {
      …
  ```
  - Deeply nested `for` and `if` blocks have high cyclomatic complexity (for example, 4 nested levels)

- Declare globals in .jshintrc:
  ```
    "globals": {
      "protractor": false,
      "browser": false,
      "element": false,
      "by": false,
      "expect": false,
      "webdriver": false,
      "$": false,
      "angular": false,
      "document": false,
      "CommonFunctions": false,
      "TestHelpers": true,
      "Utilities": true,
      "SlickGridFunctions": true,
      "PA3MainPage": true,
      "PA3EditMode": true,
      "PA3EditReportList": true,
      "PA3EditChartMode": true,
      "ChartingUtilities": true,
      "DeleteConfirmationDialog": true,
      "DocumentOptions": true,
      "TileOptions": true,
      "GroupingManager": true,
      "ModifyAccountGeneralAddtionalOptions": true,
      "ModifyAccountGeneralBasics": true,
      "ModifyAccountGeneralDates": true,
      "ModifyAccountGeneralHoldings": true,
      "ModifyAccountGeneralReturns": true,
      "ModifyAccountGips": true,
      "ModifyAccountNewPaGroupings": true,
      "ModifyAccountPaPricesBenchmark": true,
      "ModifyAccountPaAssetTypesAddRemove": true,
      "ModifyAccountPaAssetTypesSearchOrder": true,
      "ModifyAccountPaDatabases": true,
      "ModifyAccountPaFixedIncomeAnalyticsSource": true,
      "ModifyAccountPaPricesAdvanced": true,
      "ModifyAccountPaPricesPortfolio": true,
      "ModifyAccountPublisherDiscipline": true,
      "ModifyAccountPublisherBenchmarks": true,
      "ModifyAccountRiskRiskModels": true,
      "ModifyAccountRiskUniverse": true,
      "ModifyAccountCreateEditCustomGrouping": true,
      "ModifyAccountEditFee": true,
  ```

```
    "ModifyAccountNew": true,
    "DocumentOptionsPricesPortfolio": true,
    "DocumentOptionsPricesBenchmark": true,
    "DocumentOptionsPricesAdvanced": true,
    "DocumentOptionsFixedIncomeAnalyticsSource": true,
    "DocumentOptionsFixedIncomeAnalytics": true,
    "DocumentOptionsFixedIncomeAttribution": true,
    "DocumentOptionsAssetTypeAddRemove": true,
    "DocumentOptionsAssetTypeSearchOrder": true,
    "DocumentOptionsDatabases": true,
    "DocumentOptionsDates": true,
    "DocumentOptionsRiskTab": true,
    "TileOptionsDates": true,
    "TileOptionsGroupings": true,
    "CreateEditCustomGroupings": true,
    "TileOptionsColumns": true,
    "CreateEditCustomColumns": true,
    "AddNewCategory": true,
    "TileOptionsExclusions": true,
    "TileOptionsExclusionsEditGroupings": true,
    "TileOptionsExclusionsEditGroupingsAddRemove": true,
    "TileOptionsExclusionsEditGroupingsOptions": true,
    "TileOptionsHidden": true,
    "TileOptionsHiddenEditGroupings": true,
    "TileOptionsHiddenEditGroupingsAddRemove": true,
    "TileOptionsScenariosAndCashFlows": true,
    "TileOptionsRiskRiskModels": true,
    "TileOptionsUniverse": true,
    "ReportOptionsFixedIncomeTab": true,
    "AuditMode": true,
    "AnalyticsOverrideEditor": true,
    "ChangeSeries": true
  }
```

Where `true` means that it is defined in this module.

In some cases, WebStorm might also require comments for globals (else it shows error), for example:

```
/* globals CommonFunctions:true, FactSearch:true, FactSetLoginPage:true */
```

- The file must end with line feed
  Note: `gulp build` will create compiled files with LF line separators in lib directory.
- Should use "_this" to save a reference to "this"
- Add protractor as project dependency (don't assume that Jenkins has protractor)
- Don't make test script too large, else gets lint warning:
  ```
  [BABEL] Note: The code generator has deoptimised the styling of
  "C:/Users/jhwang/Documents/Source/qa-test-pa3/src/page-objects/pa3-main-page.po.js" as it
  exceeds the max of "100KB".
  ```

  **Reference:** https://www.tutorialspoint.com/software_testing_dictionary/cyclomatic_complexity.htm

## Troubleshooting

### Issue: Protractor Selenium session start exception with Chrome

Using local Selenium URL (`http://127.0.0.1:4444/wd/hub`) with webdriver-manager gets session start exception:

```
[17:19:09] E/launcher - session not created exception
from unknown error: Runtime.executionContextCreated has invalid 'context':
{"auxData":{"frameId":"8800.1","isDefault":true},"id":1,"name":"","origin":"://"}
  (Session info: chrome=54.0.2840.59)
```

The problem is the ChromeDriver.exe version 2.21 bundled with webdriver-manager. Version 2.24 does not have that issue.
To upgrade:

- Edit Protractor's config file (it does not exist by default), node_modules/protractor/config.json:
  `{"chromedriver": "2.24"}`
- Run update:
  `./node_modules/.bin/webdriver-manager update`

**Reference:** http://stackoverflow.com/questions/40100960/session-not-created-exception-for-chrome-in-protractor

## Git Commit Comment Standards

http://thief.factset.com/idiomatic-git/commit-standards.html

## WebStorm Settings

Editor > Code Style > JavaScript
Tabs and Indents:
    Tab size: 2
    Indent: 2
    Continuation indent: 4

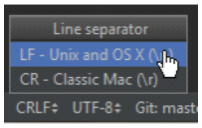
To reformat existing file, select Code > Reformat Code (Ctrl+Alt+L).

### To set up line separators for new files

- In Settings, select Editor > Code Style.
- From the Line separator (for new files) drop-down list, select the desired line separator style: **Unix and OS X (\n)**
- Apply changes and close the dialog.

### Changing line separators of existing files

The status bar (at the bottom) shows the line ending. You can change it there.



### Clear white spaces and ensure new line at end of file

- Select File > Settings
- Select Editor > General
- Uncheck Allow placement of caret after end of line
  (that will clear trailing spaces on the line on which the caret rests)
- Select All for "Strip trailing spaces on Save"
- Check "Ensure line feed at file end on Save"

### Lint

Currently project .jshintrc has this commented out, in order to disable the "deeply nested levels" warnings:

```
/* extends": "node_modules/@fds/lint-standards/jshint/.jshintrc", */
```

We will need to fix them and uncomment it.

### Module configuration, log, and other file paths

Need to be able to run in Jenkins, so don't assume pre-existing file locations outside the module folder such as C:\Tellus. That includes configuration (now using lima-override.json inside module instead of GroupID.html outside), logs, and screenshot Images folder.

Put configuration file in root of module. This will help make it modular, self-contained, and more easily accessible.

Put logs in `logs` subdirectory of module. This will help make it modular and self-contained.

Captured screenshot images are now in Images/ folder under qa-protractor-common module.

The locations can be adjusted upon feedback.

### Code Documentation

The company standard uses jsdoc.

### Reference

- http://production-applications.factset.io/libraries/publishing-libraries.html
- https://www.jetbrains.com/help/phpstorm/2016.1/configuring-line-separators.html
- http://thief.factset.com/idiomatic-git/commit-standards.html
- https://git-scm.com/book/en/v2/Git-Tools-Submodules
- https://medium.com/@porteneuve/mastering-git-submodules-34c65e940407#.2mo4q0h6t

# FAQs for Jenkins and GitLab Setup

Monday, June 05, 2017 4:40 PM

1. npm install is not installing the latest version if the latest version is not built on master
    try to move Post build script to build script
2. can we build jenkins when merge request is accepted?
    Have to try after 1. is fixed
3. gulp is not available in project root directory
    Install gulp-cli globally
4. We are unable to release v1.0 if v2.0 is already released
    Tamper package.json
5. How to delete a particular version of npm module.
    Approach developer services
6. Modules are not getting created if released from branches other than master.

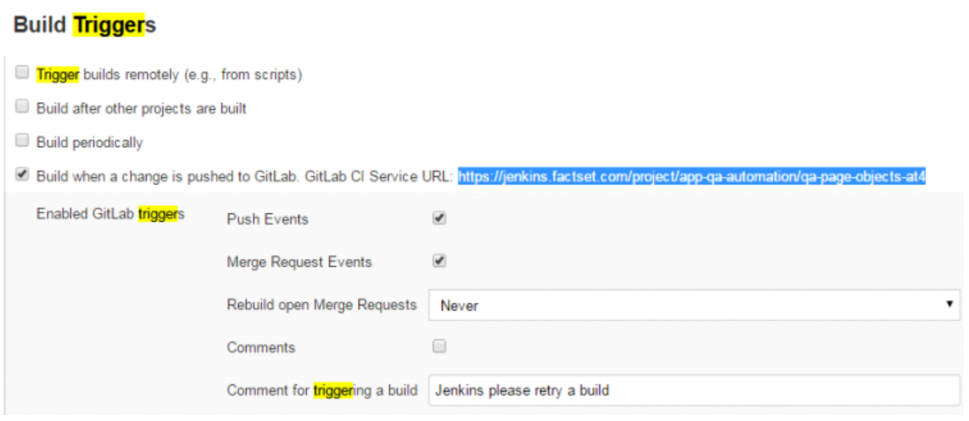
Question 1 should solve this also

# Jenkins Webhook setup

Tuesday, May 30, 2017 10:05 AM

## In Jenkins:

- Go to project, for example, https://jenkins.factset.com/job/app-qa-automation/job/qa-page-objects-at4
- Click Configuration
- Enable (webhook) trigger:



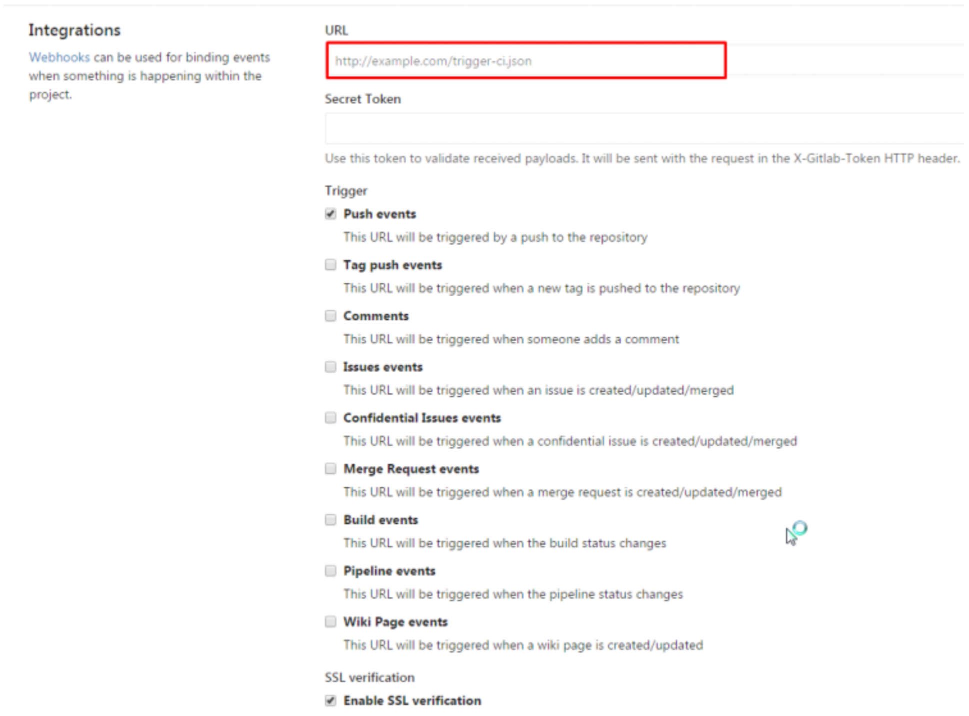
Build Triggers

- [ ] Trigger builds remotely (e.g., from scripts)
- [ ] Build after other projects are built
- [ ] Build periodically
- [x] Build when a change is pushed to GitLab. GitLab CI Service URL: https://jenkins.factset.com/project/app-qa-automation/qa-page-objects-at4

Enabled GitLab triggers

| | |
|---|---|
| Push Events | ✓ |
| Merge Request Events | ✓ |
| Rebuild open Merge Requests | Never |
| Comments | |
| Comment for triggering a build | Jenkins please retry a build |

- Copy the "GitLab CI Service URL" shown above
- Paste this URL at GITLab > Integrations Tab's URL section.



Integrations

Webhooks can be used for binding events when something is happening within the project.

URL

http://example.com/trigger-ci.json

Secret Token

Use this token to validate received payloads. It will be sent with the request in the X-Gitlab-Token HTTP header.

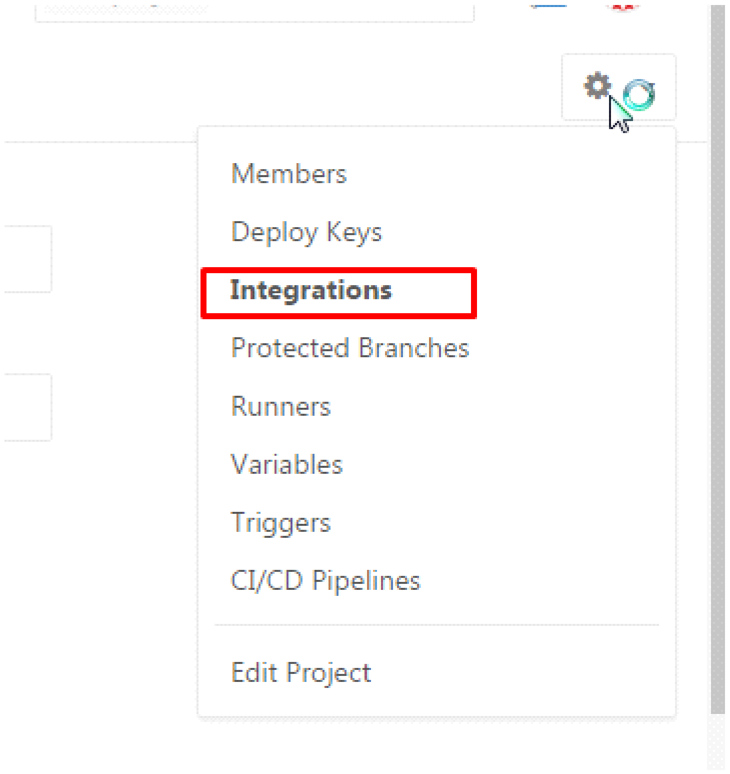
Trigger

- [x] **Push events**
  This URL will be triggered by a push to the repository
- [ ] **Tag push events**
  This URL will be triggered when a new tag is pushed to the repository
- [ ] **Comments**
  This URL will be triggered when someone adds a comment
- [ ] **Issues events**
  This URL will be triggered when an issue is created/updated/merged
- [ ] **Confidential Issues events**
  This URL will be triggered when a confidential issue is created/updated/merged
- [ ] **Merge Request events**
  This URL will be triggered when a merge request is created/updated/merged
- [ ] **Build events**
  This URL will be triggered when the build status changes
- [ ] **Pipeline events**
  This URL will be triggered when the pipeline status changes
- [ ] **Wiki Page events**
  This URL will be triggered when a wiki page is created/updated

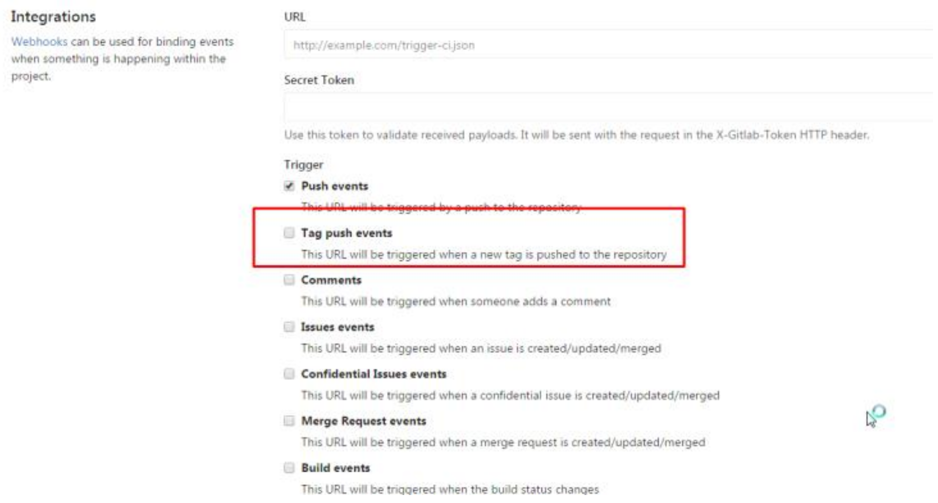
SSL verification

- [x] **Enable SSL verification**

You can visit **Integrations** from settings menu drop down:



Now select **Tag Push events** checkbox to create a webhook for the given URL.



Integrations

Webhooks can be used for binding events when something is happening within the project.

URL

http://example.com/trigger-ci.json

Secret Token

Use this token to validate received payloads. It will be sent with the request in the X-Gitlab-Token HTTP header.

Trigger

- [x] **Push events**
  ~~This URL will be triggered by a push to the repository~~
- [ ] **Tag push events**
  This URL will be triggered when a new tag is pushed to the repository
- [ ] **Comments**
  This URL will be triggered when someone adds a comment
- [ ] **Issues events**
  This URL will be triggered when an issue is created/updated/merged
- [ ] **Confidential Issues events**
  This URL will be triggered when a confidential issue is created/updated/merged
- [ ] **Merge Request events**
  This URL will be triggered when a merge request is created/updated/merged
- [ ] **Build events**
  This URL will be triggered when the build status changes

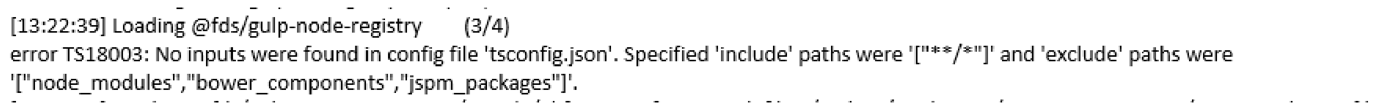
# Returning failure build for Git Projects

Tuesday, July 11, 2017    6:01 PM

| | |
|---|---|
| Subject | **Returning failure build for Git Projects** |
| From | Vasu Kodipaka |
| To | AppQA Hyderabad Automation Team |
| Sent | Tuesday, July 11, 2017 5:51 PM |

Hi Everyone,

Yesterday I have observed that builds were failing on Jenkins returning below error while releasing



```
[13:22:39] Loading @fds/gulp-node-registry      (3/4)
error TS18003: No inputs were found in config file 'tsconfig.json'. Specified 'include' paths were '["**/*"]' and 'exclude' paths were
'["node_modules","bower_components","jspm_packages"]'.
```

As suggested here : https://stackoverflow.com/questions/41211566/tsconfig-json-buildno-inputs-were-found-in-config-file

I have tried adding an empty .ts file in src folder to see if the error gets resolved

For a temporary solution, I have added the file and then it returned me a successful build

If you are seeing the above error you can try out this.

Regards,
**Vasu Kodipaka**
QA Automation Analyst II, Platform Automation
FactSet Systems India PVt Ltd
vkodipaka@factset.com
www.factset.com

FACTSET 〉SEE THE ADVANTAGE
FINANCIAL DATA | ANALYTICS | TECHNOLOGY | SERVICES

# Git Ruby GIT test development workflow

Tuesday, February 21, 2017 2:08 PM

## Example projects

GIT: https://gitlab.factset.com/app-qa-automation/fds-qa-test-financials
Jenkins: https://jenkins.factset.com/job/app-qa-automation/job/fds-qa-test-financials/

## Project naming convention

FactSet Ruby project names must start with "fds", as gem does not have namespaces. That helps to distinguish between FactSet's packages and external ones.

## Creating new project

You can the project structure from **app-qa-automation/fds-qa-test-financials** in GitHub, or generate the project using a template generation tool such as Ore below.

## Generating new project using Ore utility

**Note:** You need to do this to generate project files *only once* per project.

Create project using `mine` generator tool from `ore`:

```
gem install bundler
gem install ore
mine fds-qa-test-financials --bundler --rspec --yard
cd fds-qa-test-financials
```

(Optional) Auto-configure YARD to automatically build the yri index for installed gems:

```
$ yard config --gem-install-yri
```

You can also add the following to your .gemspec to have YARD document your gem on install:

```
# use "yard" to build full HTML docs
spec.metadata["yard.run"] = "yri"
```

See `yard config --help` for more information.

## Installing dependencies

```
bundle install
```

Running unit test:

$ rspec

Or

$ rake test

Or

$ rake

## Example Files

The lib folder has library source code. The spec folder has the tests.

### lib/fds-qa-test-financials/version.rb

By convention (of `bundle gem` and Ore), there is a version.rb in the gem project:

```
module FDSQATestFinancials
  VERSION = "0.1.8"
end
```

The version file is used by project .gemspec file.

### Gemfile

```
ruby '~> 2.3'

source 'http://artifactory.factset.com/artifactory/api/gems/rubygems/'
source 'http://artifactory.factset.com/artifactory/api/gems/rubygems-fds/'

gemspec

group :development do
  gem 'bundler', '~> 1.13'
  gem 'rake', '~>12.0'
  gem 'kramdown', '~>1.13'
  gem 'rubygems-tasks', '~>0.2'
  gem 'yard', '~>0.9'
  gem 'rspec', '~>3.5'
end
```

### fds-qa-test-financials.gemspec

```
# -*- encoding: utf-8 -*-

lib = File.expand_path('../lib', __FILE__)
$LOAD_PATH.unshift(lib) unless $LOAD_PATH.include?(lib)
```

**Additional Task before Moving to GitHub**

1. Can we come to common name for **qa-protractor-common & fds-qa-test-common?**
   **fds-qa-test-common** should probably be changed to **fds-qa-ruby-common? [Pending]**
   **Reason for this change:** If we have tool name specified in gem name it'll be easier to identify that the package belongs to Ruby. Also, we have similar naming followed for Protractor's common library.
2. In **fds-qa-test-common.rb,** use relative paths. **[Pending]**

**NOTE:** These tasks doesn't impact moving of individual projects to GitHub.

**Pending items:**

1. Eliminate the need of writing spec file for each ruby file and have single file which can call all scripts submitted to run from Tellus Environment. **[DONE]**

```
require 'fds-qa-test-financials/version'

Gem::Specification.new do |spec|
  spec.name          = 'fds-qa-test-financials'
  spec.version       = FDSQATestFinancials::VERSION
  spec.summary       = %q{Financials QA Tests}
  spec.description   = %q{Financials QA End-to-End testing}
  spec.license       = 'Nonstandard'
  spec.authors       = ['QAAutomationTeam']
  spec.email         = 'QAAutomationTeam@factset.com'
  spec.homepage      = 'https://gitlab.factset.com/app-qa-automation/fds-qa-test-financials'
  spec.files         = `git ls-files`.split($/)

  # Prevent publishing to rubygems.org
  # `metadata` requires gem v2.2
  spec.metadata['allowed_push_host'] =
'http://artifactory.factset.com/artifactory/api/gems/rubygems-fds'

  `git submodule --quiet foreach --recursive pwd`.split($/).each do |submodule|
    submodule.sub!("#{Dir.pwd}/", '')

    Dir.chdir(submodule) do
      `git ls-files`.split($/).map do |subpath|
        spec.files << File.join(submodule, subpath)
      end
    end
  end

  spec.executables   = spec.files.grep(%r{^bin/}).map{ |f| File.basename(f) }
  spec.test_files    = spec.files.grep(%r{^(test|spec|features)/})
  spec.require_paths = ['lib']

  # spec.add_development_dependency 'bundler', '~> 1.13'
  # spec.add_development_dependency 'rake', '~> 12.0'
  # spec.add_development_dependency 'kramdown', '~>1.13'
  # spec.add_development_dependency 'rubygems-tasks', '~> 0.2'
  # spec.add_development_dependency 'yard', '~> 0.9'
  # spec.add_development_dependency 'rspec', '~> 3.5'

  spec.add_runtime_dependency 'rspec', '~> 3.5'
  spec.add_runtime_dependency 'watir-webdriver', '~> 0.9'
  spec.add_runtime_dependency 'Ascii85', '~> 1.0'
  spec.add_runtime_dependency 'afm', '~> 0.2'
  spec.add_runtime_dependency 'hashery', '~> 2.1'
  spec.add_runtime_dependency 'ttfunk', '~> 1.4'
  spec.add_runtime_dependency 'childprocess', '~> 0.5'
  spec.add_runtime_dependency 'rubyzip', '~> 1.2'
  spec.add_runtime_dependency 'websocket', '~> 1.2'
  spec.add_runtime_dependency 'selenium-webdriver', '~> 2.53'
  spec.add_runtime_dependency 'mini_magick', '~> 4.3'
  spec.add_runtime_dependency 'rautomation', '~> 0.17'
  spec.add_runtime_dependency 'thread_safe', '~> 0.3'

  if (/mingw32/ =~ RUBY_PLATFORM) != nil # is win32
    spec.add_runtime_dependency 'win32-api', '~> 1.4'
    spec.add_runtime_dependency 'au3', '~> 0.1'
    spec.add_runtime_dependency 'win32-clipboard', '~> 0.6'
    spec.add_runtime_dependency 'win32screenshot', '~> 2.1'
  end
  spec.add_runtime_dependency 'tzinfo', '~> 1.2'
  spec.add_runtime_dependency 'pdf-reader', '~> 1.4'
  spec.add_runtime_dependency 'json', '~> 1.8'
  spec.add_runtime_dependency 'holidays', '~> 4.1'
  spec.add_runtime_dependency 'viewpoint', '~> 1.0'
  spec.add_runtime_dependency 'rubyXL', '~> 3.3'

end
```

### Rakefile

```
# encoding: utf-8

require 'rubygems'
require 'rake'
require 'rubygems/tasks'
Gem::Tasks.new

begin
  require 'bundler/setup'
  require 'rspec/core/rake_task'

  RSpec::Core::RakeTask.new(:spec)

  task :test    => :spec
  # task :default => %w[test]
  task :default => :spec

  require 'yard'
  YARD::Rake::YardocTask.new
  task :doc => :yard

rescue LoadError => e
  abort e.message
end

Gem::Tasks.new do |tasks|
  tasks.push.host = 'http://artifactory.factset.com/artifactory/api/gems/rubygems-fds/'
end
```

### Example test in rspec format: spec/Income Statement (RGF)/ReportSettings_spec.rb

This is optional wrapper around original Ruby test script. One can still call the Ruby test scripts directory without this.

```
lib = File.expand_path('../lib', __FILE__)
$LOAD_PATH.unshift(lib) unless $LOAD_PATH.include?(lib)

require_relative '../spec_helper'
require_relative 'ReportSettings'

describe 'ReportSettings' do
  it 'runs Report Settings' do
    RunReportSettings()
  end
end
```

It calls the original ruby test script Reportsettings.rb in the same directory.

## Installing dependencies

```
gem install bundler
bundle install
```

## Creating gem package

The **app-qa-automation/fds-qa-test-financials** project in Jenkins is configured to automatically build the gem package and publish to Artifactory when you commit to git and create a "release" tag as above.

The Jenkins job builds the gem like this:

```
bundle exec rake build
```

## Running test in GIT repository

**To run all end-to-end tests:**

```
rspec
```

**To run specific test: We are currently working on this. Idea is to avoid creating spec file for each ruby file which minimizes number of files created and maintenance overhead.**

(Windows)

```
rspec "spec\Income Statement (RGF)\Reportsettings_spec.rb"
```

(Linux)

```
rspec "spec/Income Statement (RGF)/Reportsettings_spec.rb"
```

**Please note that spec/SerialNumber.txt is using a serial number (for login) that is allocated to me for testing purposes. Please change it to your own serial number, if available. If you are using my serial number, please be aware that concurrent login with it might fail. Note: We might change the login method/configuration in the future.**

## Testing with dependent gems

You might find it easier to test with common utilities and page objects directly out of their directories without building gems for them:

```
$ rspec -I ../fds-qa-test-common/lib  -I../fds-qa-page-objects-common/lib
```

## Configuration

The out-of-the-box configuration runs tests in Selenium Grid.
To use local Chrome browser, change to this in **test-config.json**:

TestEnvironment: "FPE"

## Making a release

Note: Currently we don't have a standard tool for release, so you need to perform the following steps.

- Update the VERSION in lib/<project-name>/version.rb.
- Commit changes:
  ```
  git commit -am "chore(release) 183.0.0"
  ```
- Create a tag, for example:
  ```
  git tag 183.0.0
  ```
- Push changes to GitHub:
  (Assuming that you are in master branch)
  ```
  git push
  (If you are not in master branch)
  git push origin <branch_name>
  ```

  ```
  NOTE: Do this step immediately after pushing the changes. If there is any delay in running this
  command Jenkins will fail to release the gem to artifactory
  git push origin 183.0.0
  ```

  **Note:** The first push pushes the current branch to origin (will trigger Jenkins build). The second push pushes only the tag (without any commit, so should not trigger Jenkins build).

## Reference

- http://infonet.factset.com/view/Projects/DevServices/RubyGems
- https://www.jfrog.com/confluence/display/RTF/RubyGems+Repositories
- https://github.com/ruby-ore/ore

# Jenkins setup for building, publishing Ruby gems

Tuesday, January 03, 2017 2:37 PM

For reference check the example configuration here: https://jenkins.factset.com/job/app-qa-automation/job/fds-test-qa-financials/configure

Default Ruby version in Jenkins:

```
+ ruby --version ruby 1.8.7 (2013-06-27 patchlevel 374) [x86_64-linux]
+ gem --version 1.3.7
```

## Creating New Jenkins Project

- Click **Add New**
- **Copy from**: `examples/ruby-with-rubygems`
  (That is https://jenkins.factset.com/job/examples/job/ruby-with-rubygems/)
  Note: Must fill in "Copy from" first, else might get error
- Enter new **project name**
- Click **OK**

In new project,

- Under **Build Environment**,
- Check "**install custom tool**" option
- Select "**linux rbenv with ruby 2.2**"
- Click **Save**

## Jenkins Configuration: Build -> Execute Shell

```
eval "$(rbenv init -)" || true
rbenv install -s 2.3.3
rbenv shell 2.3.3

export GEM_HOME=$(pwd)/.gem
export PATH="$PATH:$GEM_HOME/bin"

gem install bundler
bundle install
bundle exec rake build
ls pkg/

export RUBYGEMS_HOST=http://artifactory.factset.com/artifactory/api/gems/rubygems-fds/
version=`ruby -I./lib -r 'fds-qa-test-financials/version' -e 'puts FDSQATestFinancials::VERSION'`
#version=$(git tag -l --points-at HEAD)

if [ ! -z $version ]; then
    gempath=pkg/fds-qa-test-financials-$version.gem
    gem push $gempath --host http://artifactory.factset.com/artifactory/api/gems/rubygems-fds
fi
```

## Background information

To add the artifactory the mirror of rubygems.org, run the following command.

```
gem source -a http://artifactory.factset.com/artifactory/api/gems/rubygems/
```

To add the factset internal gems run:

```
gem source -a http://artifactory.factset.com/artifactory/api/gems/rubygems-fds/
```

### With Bundler

Be sure to replace the rubygems.org source with our artifactory mirror in your Gemfile. Existing projects will need to regenerate their Gemfile.lock.

```
source "https://rubygems.org"
```

Should become

```
source  "http://artifactory.factset.com/artifactory/api/gems/rubygems/"
```

If you need FactSet internal gems be sure to add the rubygems-fds source:

```
source  "http://artifactory.factset.com/artifactory/api/gems/rubygems-fds/" do
    gem "fds_foo", "~> 1.1.0"
    gem "FDSbar", "~> 0.10.0"
end
```

Older versions of bundler do not support the source block dsl, you may instead add the source line to the top of your file, or individually on the gems that require it.

```
source http://artifactory.factset.com/artifactory/api/gems/rubygems-fds/"
gem "fds_foo", "~> 1.1.0"
```

Or,

```
gem "FDSbar", "~> 0.10.0", :source => "http://artifactory.factset.com/artifactory/api/gems/rubygems-fds/"
```

See the http://bundler.io/gemfile.html bundler docs for more details.

If you are using either of the factset provided rubygems mirrors in your jenkins project, good news! Jenkins is already configured to use the FactSet rubygems mirror and the internal packages repository. However, you should have your project configured to use bundler with the two factset sources as described in the With Bundler section.

If you maintain a ruby gem and would like to push artifacts to Artifactory you will run:

```
gem push <PACKAGE> --host http://artifactory.factset.com/artifactory/api/gems/rubygems-fds
```

We have setup an example jenkins job, for your convience. This job will automatically push your gem to artifactory when a new git tag is on master.

If you are using **rubygems 2.2** or later, we recommend that you set the "allowed_push_host" setting in your Gemspec to avoid accidental pushes to the public rubygems repository.

```
Gem::Specification.new 'my_gem', '1.0' do |s|
  # ...
  s.metadata['allowed_push_host'] = 'http://artifactory.factset.com/artifactory/api/gems/rubygems-fds'
end
```

## Naming policy

**Packages authored at FactSet for internal use must have their names start with fds (case insensitive).**
Because rubygems does not support namespacing, this is the only way to prevent namespace conflicts with public rubygems packages.

## Reference

http://infonet.factset.com/view/Projects/DevServices/RubyGems

https://www.jfrog.com/confluence/display/RTF/RubyGems+Repositories

# Executing tests with Ruby gems

Monday, July 31, 2017 9:29 AM

## QA Tests gem installation

```
export RUBYGEMS_HOST=http://artifactory.factset.com/artifactory/api/gems/rubygems-fds/

export GEM_HOME=$(pwd)/.gem
export PATH="$PATH:$GEM_HOME/bin"

gem install fds-qa-test-financials
```

## Executing tests in Ruby gem

(Linux)

```
#!/bin/bash
dir=$(ruby -e "puts Gem::Specification.find_by_name('fds-qa-test-financials').gem_dir")
rspec --default-path $dir/spec --options $dir/regression.spec
```

(Windows BATCH)

```
@ECHO OFF
for /f %%i in ('ruby -e "puts Gem::Specification.find_by_name('fds-qa-test-financials').gem_dir"') do set dir=%%i
rspec --default-path %dir%\spec --options %dir%\regression.spec
```

The `for` statement sets `dir` environment variable to the installation directory of the fds-qa-test-financials gem. The rspec statement runs the tests.

# QA tests as sub-project (GIT submodule)

Tuesday, December 06, 2016 7:48 AM

Suppose that `qa-test-analytics` is a "super-project" containing sub-project `qa-test-pa3`.

**Notation:** $ is command line prompt.

## Adding submodule

To add qa-test-pa3 as a sub-project:

```
$ git submodule add git@gitlab.factset.com:app-qa-automation/qa-test-pa3.git
```

Commit the fact that we added a submodule:

```
$ git commit -am 'chore(submodule) added qa-test-pa3 module'
```

Push changes to gitlab:

```
$ git push -u origin master
```

Note: If you are not on master branch, use the that branch's name.

## Cloning a project with submodules

To clone a super-project with submodules:

```
$ git clone --recursive git@gitlab.factset.com:app-qa-automation/qa-test-
analytics.git
```

The `--recursive` option will automatically "init" (initialize) and "update" (download content of) the sub-projects.

> **Note:** If you don't use `--recursive` option, then you, will need to do this manually:
>
> ```
> $ git clone git@gitlab.factset.com:app-qa-automation/qa-test-
> analytics.git
> $ git submodule init
> $ git submodule update
> ```

## Fetching sub-project updates

To fetch sub-project updates made by other developers:
(under super-project root or subdirectory)

```
$ git submodule update --remote --merge
```

That will fetch and try to merge the changes for all sub-projects.

To update only one sub-project:

```
$ git submodule update --remote --merge qa-test-pa3
```

**Note:** You can also do that manually:

```
$ git fetch
$ git merge origin/master
```

**Note:** If you forget the `--merge`, Git will just update the submodule to whatever is on the server and reset your project to a detached HEAD state.

```
$ git submodule update --remote
Submodule path 'DbConnector': checked out
'5d60ef9bbebf5a0c1c1050f242ceeb54ad58da94'
```

If this happens, don't worry, you can simply go back into the directory and check out your branch again (which will still contain your work) and merge or rebase origin/stable (or whatever remote branch you want) manually.

## Working on sub-project

- First go into sub-project directory:

  ```
  $ cd qa-test-pa3
  ```

- Check out a branch:

  ```
  $ git checkout qa
  ```

- Merge upstream changes:

  ```
  $ git submodule update --remote --merge
  ```

- Make your changes
- Commit your changes:

  ```
  $ git commit -am 'docs(README) add unicode support'
  ```

- Merge upstream changes again and check that there is no conflict:

  ```
  $ git submodule update --remote --merge
  ```

  If it shows any conflict, fix the conflict, then commit and submodule update again.

- When merge is successful, commit and push changes upstream:
  (under sub-project directory)

  ```
  $ git commit -am "chore(qa-test-pa3) merge with upstream"
  $ git push -u origin
  ```

- You also need to commit the fact that a submodule changed in the *super*-project:
  - First cd back up to *super*-project:

    ```
    $ cd ..
    ```

  - Commit the submodule changes in the super-project:

    ```
    $ git commit -am "chore(qa-test-pa3) commit qa-test-pa3 module changes"
    ```

- And push upstream:

  ```
  $ git push -u origin
  ```

  **Note:** You need to do this, in order for clones of the *super*-project to see the submodule changes.

## See Also

Test script development workflow:
Test Development Git Workflow for JavaScript/Protractor

### Reference

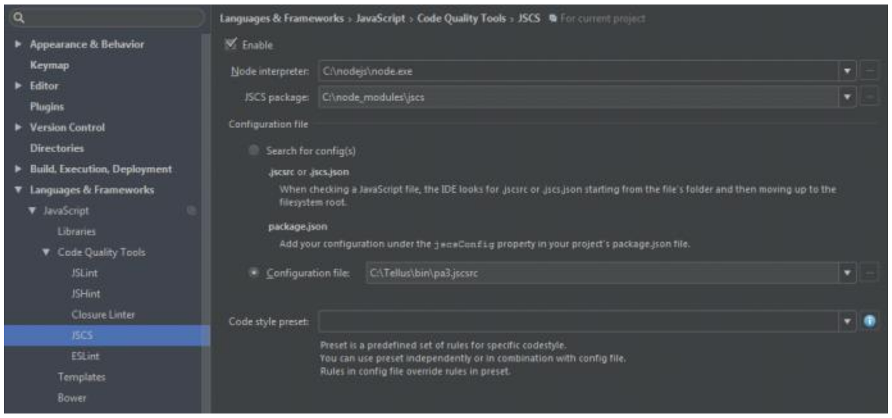
https://medium.com/@porteneuve/mastering-git-submodules-34c65e940407#.2mo4q0h6t

https://git-scm.com/book/en/v2/Git-Tools-Submodules

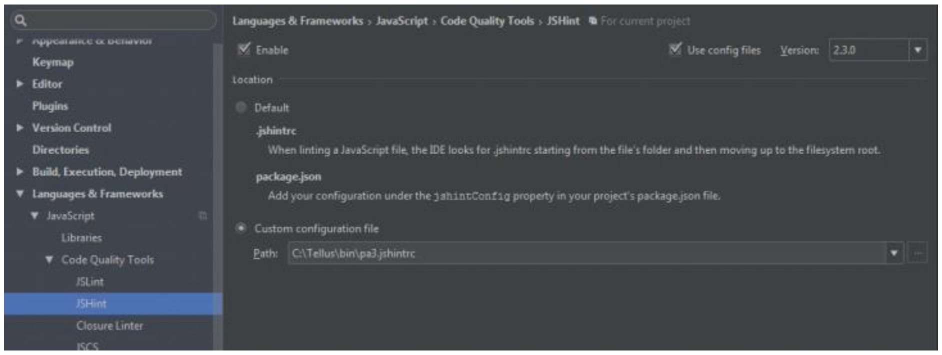
# Lint standards setup

Thursday, February 02, 2017 4:44 PM

Follow below steps to set up @fds/lint-standards and to wire the module with Webstorm IDE:

- Ensure **FDS registry setup** is complete. To complete the set up follow instructions mentioned here.
- Install @fds/lint-standards on C: drive if your Webstorm is consuming modules locally for your project.
  Run the command **npm install @fds/lint-standards** on the command prompt.
- Install the JSCS module.
  Run the command **npm install jscs** on the command prompt.
- Navigate to the installed location C:\node_modules\@fds\lint-standards\jscs\presets and open factset.json in edit mode.
- Remove "esnext" as a dependency and save the file. Note: Setting it to 'false' doesn't work, remove the line and save the file.
- Fetch the appropriate .jscsrc file from your respective product's repository and have it placed in the **bin** folder in the root directory of your project.
- Fetch the .jshintrc file from the //depot/smqa/Tellus/ProtractorDev/QAClass/.jshintrc and have it placed in the **bin** folder in the root directory of your project.
- In the Webstorm IDE, open the Settings dialog.
- Navigate to Languages & Frameworks-> JavaScript-> Code Quality Tools-> JSCS
- Enable the JSCS tool and hook it up with the node modules location and the Node interpreter appropriately.
- Select the 'Configuration file' radio button option and browse for the .jscsrc file in the bin folder.
  If node modules are setup on C drive, your dialog should look as below.
- 

  Languages & Frameworks › JavaScript › Code Quality Tools › JSCS For current project
  ☑ Enable
  Node interpreter: C:\nodejs\node.exe
  JSCS package: C:\node_modules\jscs
  Configuration file
  ○ Search for config(s)
  .jscsrc or .jscs.json
  When checking a JavaScript file, the IDE looks for .jscsrc or .jscs.json starting from the file's folder and then moving up to the filesystem root.
  package.json
  Add your configuration under the jscsConfig property in your project's package.json file.
  ● Configuration file: C:\Tellus\bin\pa3.jscsrc
  Code style preset:
  Preset is a predefined set of rules for specific codestyle.
  You can use preset independently or in combination with config file.
  Rules in config file override rules in preset.

- Navigate to Languages & Frameworks-> JavaScript-> Code Quality Tools-> JSHint
- Enable the JSHint tool and select the 'Custom configuration file' radio button and browse for the .jshintrc file in the bin folder.
- 

  Languages & Frameworks › JavaScript › Code Quality Tools › JSHint For current project
  ☑ Enable ☑ Use config files Version: 2.3.0
  Location
  ○ Default
  .jshintrc
  When linting a JavaScript file, the IDE looks for .jshintrc starting from the file's folder and then moving up to the filesystem root.
  package.json
  Add your configuration under the jshintConfig property in your project's package.json file.
  ● Custom configuration file
  Path: C:\Tellus\bin\pa3.jshintrc

- Apply the changes to get started.

Fixing lint errors:

- Once Webstrom is wired to the .jshintrc and .jscsrc files, fixing the lint errors is straight forward.
- Open the file and wait for Webstorm plugins to find the errors.
- Select the entire content of the file, right click and select the "Fix JSCS Problems".

Note: 'Fix JSCS Problems' options is available with latest versions of Webstorm only. The logic issues like Cyclomatic Comple xity, 'Too deeply nested' etc. will still need to be cleaned up manually.

Additional Notes:

- It is recommended the user configure Webstorm's Code Style tools with the necessary parameters as per the project standards.
- Navigate to on the settings dialog to Editor -> Code Style -> JavaScript to explore.
- Once the configuration is done, you may select the file content and do a 'Reformat Code'. This can be found from the **Code** menu from the menu bar. You may also use the shortcut **CTRL+ALT+L**.

# QA e2e Testing Guide for Engineers

Tuesday, November 08, 2016 10:36 AM

## Introduction

QA tests are available as NPM packages from Artifactory, and they are built from source in GitHub. Basic usage looks like this:

```
npm install @fds/qa-test-us
./node_modules/.bin/protractor ./node_modules/@fds/qa-test-us/cid.conf.js
```

More detailed information follows.

## Prerequisite

You need to configure @fds in npm first:

```
npm config set registry http://artifactory.factset.com/artifactory/api/npm/npm/

npm config set @fds:registry http://artifactory.factset.com/artifactory/api/npm/npm-fds/
```

## NPM Package Usage

The QA tests are published as NPM package (e.g. `@fds/qa-test-pa3)` in Artifactory npm repository server. It is designed to be ready to run "out-of-the-box", as much as possible.

To install test package:

```
npm install @fds/qa-test-us
```

## Package.json Configuration

In your package.json, you could add QA test package as dependency. The npm `--save-dev` option will save dev dependencies in your package.json. For example,

```
"devDependencies": {
  "@fds/qa-test-us": "^76.1.0"
}
```

## Serial Number API

Some QA e2e tests (such as Universal Screening) need Serial Number to run. Tellus Serial Number REST API can allocate serial number and set FDSAv2 override. For example, you can call it using PostMan.

### Example Serial Number API Request:

POST http://tellusstgb01:8080/tellusci/ci/request

```
{
  "RequestJson": {
    "serialnumber": "random",
    "vmsusername": "FDSQAR_C",
    "fdsav2stage": "edge",
    "fdsav2overrides": "review_universal-screening_version-0-1_2248"
  }
}
```

### Example Response:

```
{
    "ResponseJson": {
        "requestId": 2018010510481098372,
        "serialnumber": 686362,
        "success": "Successfully blocked serial number"
    }
}
```

The "random" value for serialnumber will allocate any serial number.  Enter the acquired serial number in test-config.json below.

Remember to free the serial number when you are done.

### Example Release of Serial Number:

POST http://tellusstgb01:8080/tellusci/ci/release

```
{"RequestJson": {"serialnumber": 686362}}
```

### Example Response for Release:

```
{
    "ResponseJson": {
        "requestId": 2018010510500002463,
        "serialnumber": 686362,
        "success": "Successfully released serial number"
    }
}
```

## Running tests

The QA e2e tests use configuration files called test-config.json and SerialNumber.txt in the current directory.  Create `test-config.json` configuration file in your current working directory.  Enter the serial number above in test-config.json.

Example test-config.json:

```
{
  "Stage": "edge",
  "FDSAv2Stage": "edge",
  "FDSAv2Overrides": "review_universal-screening_moc-breakautomation_2426",
  "HTMLReports": "Staging",
  "PrebuildName": "",
  "SGBrowserName": "chrome",
  "SGBrowserVersion": "61",
  "VMSUserID": "FDSQAR_C",
  "SerialNumber": "686363",
  "UserName": "automation.tc686363",
  "Password": "TellusAut0mation",
  "TestEnvironment": "SGE",
  "SeleniumGridHub": "http://tellusdevb03.pc.factset.com:4444/wd/hub"
}
```

To test with Prebuild called "afetterman.at4", for example, use this:

```
  "Stage": "Prebuild",
  "PrebuildName": "afetterman.at4",
```

## Running tests

To run test (with test-config.json above):

(Linux command syntax)

```
npm install @fds/qa-test-us
./node_modules/bin/protractor node_modules/@fds/qa-test-us/cid.conf.js
```

Note: On the command line, you can also override the baseUrl, which is in the conf.js with `--baseUrl` option, for example,

```
./node_modules/.bin/protractor --baseUrl https://universalscreening.apps.factset.com node_modules/
@fds/qa-test-us/cid.conf.js
```

## GitHub Repo

The QA test repositories are generally in GitHub.  In that case you need to access it:

```
git clone git@github.factset.com:app-qa-automation/qa-test-us.git
cd qa-test-us
```

## Reference

- http://production-applications.factset.io/libraries/publishing-libraries.html
- https://www.jetbrains.com/help/phpstorm/2016.1/configuring-line-separators.html
- http://thief.factset.com/idiomatic-git/commit-standards.html
- https://git-scm.com/book/en/v2/Git-Tools-Submodules
- https://medium.com/@porteneuve/mastering-git-submodules-34c65e940407#.2mo4q0h6t

# Embedding QA test project as sub-project (Optional)

Monday, September 11, 2017 7:24 AM

If you have the need to update the QA test scripts, you are encouraged to fork the QA project in Gitlab, and submit Merge Requests. However, an optional alternative is embedding QA test project as sub-project under your project, but that takes more effort and is more prone to mistakes.

Suppose that `qa-test-analytics` is a super-project containing sub-project `qa-test-pa3`, for example.

Please see test script development workflow:
Test Development Git Workflow for JavaScript/Protractor

**Notation:** $ is command line prompt.

## Adding submodule

**To add qa-test-pa3 as a sub-project:**

```
$ git submodule add git@gitlab.factset.com:app-qa-automation/qa-test-pa3.git
```

Commit the change:

```
$ git commit -am 'chore(submodule) added qa-test-pa3 module'
```

Push to gitlab:

```
$ git push -u origin master
```

Note: If you are not on master branch, use the that branch's name.

## Cloning a project with submodules

To clone a super-project with submodules:

```
$ git clone --recursive git@gitlab.factset.com:app-qa-automation/qa-test-
analytics.git
```

The `--recursive` option will automatically "init" (initialize) and "update" (download content of) the sub-projects.

**Note:** If you don't use `--recursive` option, then you, will need to do this manually:

```
$ git clone git@gitlab.factset.com:app-qa-automation/qa-test-
analytics.git
$ git submodule init
$ git submodule update
```

## Fetching sub-project updates

To fetch sub-project updates made by other developers:
(under super-project root or subdirectory)

```
$ git submodule update --remote --merge
```

That will fetch and try to merge the changes for all sub-projects.

To update only one sub-project:

```
$ git submodule update --remote --merge qa-test-pa3
```

> **Note:** You can also do that manually:
>
> ```
> $ git fetch
> $ git merge origin/master
> ```

> **Note:** If you forget the `--merge`, Git will just update the submodule to whatever is on the server and reset your project to a detached HEAD state.
>
> ```
> $ git submodule update --remote
> Submodule path 'DbConnector': checked out
> '5d60ef9bbebf5a0c1c1050f242ceeb54ad58da94'
> ```
>
> If this happens, don't worry, you can simply go back into the directory and check out your branch again (which will still contain your work) and merge or rebase origin/stable (or whatever remote branch you want) manually.

## Working on sub-project

- First go into sub-project directory:

  ```
  $ cd qa-test-pa3
  ```

- Check out a branch:

  ```
  $ git checkout qa
  ```

- Merge upstream changes:

  ```
  $ git submodule update --remote --merge
  ```

- Make your changes
- Commit your changes:

  ```
  $ git commit -am 'docs(README) add unicode support'
  ```

- Merge upstream changes again and check that there is no conflict:

  ```
  $ git submodule update --remote --merge
  ```

  If it shows any conflict, fix the conflict, then commit and submodule update again.

- When merge is successful, commit and push changes upstream:
  (under sub-project directory)

  ```
  $ git commit -am "chore(qa-test-pa3) merge with upstream"
  $ git push -u origin
  ```

- You also need to commit the fact that a submodule changed in the *super*-project:
    - First cd back up to *super*-project:

      ```
      $ cd ..
      ```

    - Commit the submodule changes:

      ```
      $ git commit -am "chore(submodule) commit qa-test-pa3 module changes"
      ```

    - And push upstream:

      ```
      $ git push -u origin
      ```

      **Note:** You need to do this, in order for clones of the *super*-project to see the submodule changes.

## Running tests in submodules

This will run all tests in all submodules:

```
$ git submodule foreach 'npm test'
```

Substitute `npm test` with any other testing command.

# CID Pipeline

Monday, September 11, 2017 7:32 AM

## QA CID Pipeline Jenkins Project:

qa-cid-test

## Pipeline script repo:

app-qa-automation / qa-cid-test

## Jenkinsfile

```
#!groovy
node('cid') {
stage('Initialize') {
        echo 'Initializing...'
    }
stage('Checkout') {
        echo 'Getting source code...'
        checkout scm
    }
stage('Build') {
        echo 'Installing dependencies...'
    }
stage('Test') {
        echo 'Testing...'
// Example CID data
        map = [
                "app-name": "cid-test-service",
                "repo-url": "git@gitlab.factset.com:bbatha/cid-test-
service.git",
                "commit": "8428dccb55db6991687a755bb6d5e5bcef90d39d",
                "factset-json": """{
                "name": "cid-test-service",
                "owners": {
                    "page": "cid_oncall",
                    "admin_email": "cid-admin@factset.com"
                },
                "type": "service",
                "scripts": {
                    "test": "npm test"
                },
                "config_version": 0.2,
                "deployment": {
                    "factsetio": {
                        "buildpacks": [{
                                                   "url":
"https://github.com/ryandotsmith/null-buildpack"
                                               },
                                               {
                                                   "url":
"http://artifactory.factset.com/artifactory/repo/legacy-node-platform-
buildpack/legacy-node-platform-buildpack.tgz"
                                               }
                        ],
                        "formation": {
                            "web": {
                                "quantity": 2,
                                "size": "1X"
                            }
                        }
                    }
                }
            }""",
                "fdsav2-url": "cid-test-service.services.nprod.factset.com",
                "e2e-label": "e2e_cid-test-service_4f8fce64-02ad-4eef-
bba5-81fb4782f90b",
                "merge-request-id": "83357",
                "repo-id": "bbatha/cid-test-service"
        ]
def testRunner = load "TestRunner.groovy"
        testRunner.RunTestsForApp(map)
}
/*
    stage('Publish') {
        echo 'Publishing Test Coverage...'
        publishHTML (target: [
                allowMissing: false,
                alwaysLinkToLastBuild: false,
                keepAll: true,
                reportDir: 'coverage/lcov-report',
                reportFiles: 'index.html',
                reportName: "Application Test Coverage"
        ])
    }
    */
}
```

## TestRunner.groovy:

```
import groovy.json.JsonSlurper
/**
 * Get version for stage from Tracker API.
 * @param appName
 * @param stage default "devel"
 * @param debug true for debug output
 * @return
 */
def getVersion(appName, stage="devel", debug=false) {
  /*
    Example response JSON:
    {"actions":[{"label_name":"online_2017_07_11_001","cluster":"fxdev1-
a","stage":"devel","label_minor":5,"status":1,"message":null,"stamp":"2017-07
-11
18:41:10-04","user":null,"label_id":2736,"platform":"Fonix","id":141857,"type
":"promote","patch":2,"label_version":203,"overwritten":0}],"last_stamp":"201
7-07-11 18:41:10-04","status":1,"last_status":1}
   */
  def trackerUrl = "http://tracker.factset.com/build_status?stage=
${stage}&type=promote"
  def response = trackerUrl.toURL().text
  echo(response)
  def jsonSlurper = new JsonSlurper()
  def json = jsonSlurper.parseText(response)
  def actions = json.actions
  def rv = null
  if (actions.size() > 0) {
    def item = actions[0]
    rv = item.label_version
  }
  return rv
}
def RunTestsForApp(map) {
  // app-name string Name of the app
  // repo-url string URL to the git repo
  // commit string The commit being built
  // factset-json map The factset.json of the app
  // fdsav2-url string The FDSAv2 URL for the app. i.e;
<appname>.nprod.services.factset.com
  // e2e-label string The label generated for holding the deployment to be
tested

  echo "RunTestForApp with "
  for (e in map) {
    echo "${e.key}: ${e.value}"
  }
// Initialize environment
  def node = tool name: 'node 6.x.x LTS', type:
'jenkins.plugins.nodejs.tools.NodeJSInstallation'
  env.PATH = "${node}/bin:${env.PATH}"
sh """
    node --version
    npm --version
  """
//  def repoUrl = map["fdsav2-url"]
  def repoUrl = "pru.staging-cauth.factset.com"
//  def appName = map["app-name"]
  def appName = "qa-test-pru"
  def versionSuffix = getVersion(appName)
  echo "Tracker version: ${versionSuffix}"
  if (versionSuffix != null) versionSuffix = "@^" + versionSuffix + ".0.0"
  def cfDir = '.'
def json="""
    {
      "Stage": "Devel",
      "FDSAv2Stage": "Devel",
      "FDSAv2Overrides": "-1",
      "HTMLReports": "Staging",
      "PrebuildName": "",
      "BrowserType": "Chrome",
      "BrowserVersion": "54",
      "VMSUserID": "FDSQAR_C",
      "SerialNumber": "381970",
      "UserName": "automation.tc381970",
      "Password": "TellusAut0mation",
      "TestEnvironment": "SGE"
    }
    """

  writeFile file: "test-config.json", text: json

  try {
    sh """
      npm prune
      npm install --global-style --no-shrinkwrap --no-package-lock
protractor@^5.1.2
      ./node_modules/.bin/protractor --version
      npm install --no-shrinkwrap --no-package-lock @fds/
${appName}${versionSuffix}
      ./node_modules/.bin/protractor --params.path=${cfDir} --
baseUrl=https://${repoUrl}/ node_modules/@fds/qa-test-pru/regression.conf.js

    """
} catch (e) {
    emailext (
      to: "jhwang@factset.com",
      subject: "FAILED: Job '${env.JOB_NAME} [${env.BUILD_NUMBER}]'",
      mimeType: "text/html",
      body: """<p>FAILED: Job '${env.JOB_NAME} [${env.BUILD_NUMBER}]':</p>
         <p>Check console output at &QUOT;<a
href='${env.BUILD_URL}'>${env.JOB_NAME}  [${env.BUILD_NUMBER}]
</a>&QUOT;</p>""",
      recipientProviders: [[$class: 'DevelopersRecipientProvider']]
    )
    throw e
  }

  echo "Tests complete!"
}

return this
```

# Git Training Introduction

Friday, July 11, 2014 10:31 AM

It is easier and preferable to use TortoiseGit and https instead of SSH.

## Git Guides

- Interactive tutorial: https://try.github.io
- Free online version of the *Pro Git* book: http://git-scm.com/book
- http://pcottle.github.io/learnGitBranching/
- GitHub Learning Resources
- Git Immersion
- git - the simple guide

## FactSet Gitlab Training and Documentation

- https://gitlab.factset.com/bchaiklin/web-training/tree/master/presentations/git
- http://infonet.factset.com/view/Projects/DevServices/GitLab
- http://infonet.factset.com/view/Main/TransitionToGitlab

# Git Terminology

Tuesday, July 15, 2014 8:46 AM

## Terminology

*master(master  branch)*

Trunk; the main line

*Staged*

Staged files are files we have told Git that are ready to be committed

-- (dash)

For example: `git diff -- HEAD`

When an argument can be misunderstood as either a revision or a path, they can be disambiguated by placing -- between them. E.g. `git diff -- HEAD` is, "I have a file called HEAD in my work tree. Please show changes between the version I staged in the index and what I have in the work tree for that file", not "show difference between the HEAD commit and the work tree as a whole". You can say `git diff HEAD --` to ask for the latter.

# Gitlab/NPM Quick Start: PA3 test script repository

Tuesday, November 08, 2016 10:36 AM

## Quick Start for Consumers of PA3 QA Tests

PA3 test script repository:

https://gitlab.factset.com/app-qa-automation/qa-test-pa3

Protractor common utilities (used by qa-test-pa3 and others):

https://gitlab.factset.com/app-qa-automation/qa-protractor-common

Example Jenkins project running regression test:

https://jenkins.factset.com/job/app-qa-automation/job/test-only-example/

### Prerequisite

You need to setup @fds in npm first (both producer and consumer of npm packages):

```
npm config set registry
http://artifactory.factset.com/artifactory/api/npm/npm/

npm config set @fds:registry
http://artifactory.factset.com/artifactory/api/npm/npm-fds/
```

### Gitlab Repo Quick Start

```
git clone git@gitlab.factset.com:app-qa-automation/qa-test-pa3.git

cd qa-test-pa3

npm install

npm run regression
```

### NPM Quick Start

The test script repository is published as NPM package(`@fds/qa-test-pa3` in Artifactory. It includes regression test package with tag 'regression', @fds/qa-test-pa3@regression, which is designed to run "out-of-the-box" on machines that have T: drive mapping (such as some App QA Automation laptops and FocalPoint Environment QA VMs).

**Note:** You can check if you have `T:` drive mapping by executing the following in Windows Command Prompt:

```
$ net use T:
Local name        T:
Remote name       \\tellusdevb01.pc.factset.com\vms\appqa_vms\FDSQAR_C-123456
Resource type     Disk
Status            OK
# Opens           0
# Connections     1
The command completed successfully.
```

The Serial Number would be `123456` after `FDSQAR_C-` in `Remote name`.

Simple usage on machines with `T:` drive:

**Linux:**
```
npm install --save-dev @fds/qa-test-pa3@regression

node_modules\/bin/protractor node_modules/\@fds/qa-test-
pa3/regression.conf.js
```

**Windows:**
```
npm install --save-dev @fds/qa-test-pa3@regression

.\node_modules\.bin\protractor node_modules\@fds\qa-test-pa3
\regression.conf.js
```

**Note:** The `--save-dev` will save dev dependencies in your package.json, which could use the npm dist-tag **`"regression"`** as the version range:

```
  "devDependencies": {
    "@fds/qa-test-pa3": "regression"
  }
```

On other development workstations and Jenkins, you need to configure using `test-config.json` file or command line parameters (--params.NAME=VALUE) to configure the product SerialNumber. We will describe test-config.json below. Please see https://gitlab.factset.com/app-qa-automation/qa-test-pa3 for more information on command line parameters.

Create `test-config.json` configuration file in your current working directory. Typically that is your Gitlab project root or Jenkins project root.
Example test-config.json:
```
{
  "Stage": "Prebuild",
  "PrebuildName": "afetterman.at4",
  "SerialNumber": 123456
}
```

The SerialNumber must be a valid product serial number assigned to you.
Then you can execute the same protractor command above to run tests.

The included regression test is compatible to "Live" stage, that is, the regression test (``npm run regression`` configured in the repository's package.json) uses this default configuration:
```
Stage: 'live'
HTML Reports: 'production'
seleniumAddress: "http://tellusdevb03.pc.factset.com:4444/wd/hub", // Web
Production
browserName: 'chrome',
Version: '47'
```

To test against another Stage, for example, a Prebuild called "afetterman.at4", create this `test-config.json` in your project root:

```
{
  "Stage": "Prebuild:afetterman.at4",
  "SerialNumber": 123456
}
```

or equivalently,

```
{
  "Stage": "Prebuild",
  "PrebuildName": "afetterman.at4",
  "SerialNumber": 123456
}
```

Note: The property names and values are case-insensitive.

## Configuring your `run regression` command

You could configure your `package.json` to use the regression test package like this:

```
{
  "scripts": {
    "regression": "./node_modules/.bin/protractor ./node_modules/@fds/qa-test-pa3/regression.conf.js"
  },
}
```

Then you can run regression test like this:

```
npm run regression
```

Note: This NPM package is configured to install protractor as dependency, so you don't need to separately install protractor.

## Testing Against Different Web Application

If you need to test against your own version of Web application instead of the FactSet staging server, you can change the URL in node_modules/@fds/qa-test/regression.conf.js:

```
  baseUrl: 'https://localhost/',
```

## Jenkins Job Example

https://jenkins.factset.com/job/app-qa-automation/job/test-only-example/

That job is configured to run regression test very night.  It consumes NPM package, without using Gitlab.

**Example Jenkins job script:**

```
# store temporary files within the workspace to avoid filling /tmp and to avoid name collisions between instances of build tools
export TMPDIR="${WORKSPACE}/tmpdir"

# build project in production mode (https://gitlab.factset.com/developer-services/developer-services/blob/master/workflows/builds.md#build-modes)
export GULP_ENV=development

json='
{
  "stage": "qa",
  "htmlReports": "staging",
  "serialNumber": 123456
}'

echo "$json" > test-config.json
npm install @fds/qa-test-pa3@regression
./node_modules/.bin/protractor node_modules/\@fds/qa-test-
pa3/regression.conf.js
```

**Note**: Please use valid SerialNumber assigned to you.

## Tellus Cron Job Usage of Test Packages

Tellus Cron job would consume packages with dist-tags `live`, `qa`, `devel` for  the respective Stages.  For example, if user selects "QA" for Test Case Location (a.k.a. "Test Case Stage"), Tellus would download test scripts like this:

```
npm install @fds/qa-test-pa3@qa
```

# npm Artifactory @fds setup

Tuesday, October 31, 2017 8:12 AM

In order to use Artifactory, you need to configure @fds in npm first:

```
npm config set registry http://artifactory.factset.com/artifactory/api/npm/npm/

npm config set @fds:registry http://artifactory.factset.com/artifactory/api/npm/npm-fds/
```

To check:

npm config list

It should show:

```
@fds:registry = "http://artifactory.factset.com/artifactory/api/npm/npm-fds/"
http://artifactory.factset.com/artifactory/api/npm/npm-fds/ = ""
...
registry = "http://artifactory.factset.com/artifactory/api/npm/npm/"
```

# FDS registry setup

Tuesday, November 08, 2016 1:59 PM

FDS registry setup should be done for both the producer and consumer of npm packages in Artifactory.

```
npm config set registry
http://artifactory.factset.com/artifactory/api/npm/npm/

npm config set @fds:registry
http://artifactory.factset.com/artifactory/api/npm/npm-fds/
```

The first changes the default unscoped registry. The second changes @fds scoped registry.

To confirm npm has been configured with the new registry URLs run,

```
npm config list
```

You should see output similar to:

```
; userconfig /home/user/hbetts/.npmrc
@fds:registry = "http://artifactory.factset.com/artifactory/api/npm/npm-fds/"
registry = "http://artifactory.factset.com/artifactory/api/npm/npm/"
```

## Reference

- http://production-applications.factset.io/getting-started/node.html
- http://production-applications.factset.io/libraries/publishing-libraries.html
- http://infonet.factset.com/view/Projects/DevServices/Artifactory

For installing private ruby gems you have to register the following:
Gem source --add http://artifactory.factset.com/artifactory/api/gems/rubygems/
Gem source --add http://artifactory.factset.com/artifactory/api/gems/rubygems-fds/

# Getting Started with Jenkins

Friday, October 14, 2016 2:52 PM

The Jenkins instance managed by Developer Services is located at https://jenkins.factset.com

- If your group is using Jenkins for the first time, file an RPD and fill out the questions in the form to request a group folder.
  If you would like to create a job in an existing folder/group, you should contact the owner of your folder (so they can create it or grant you permissions to); if you are unsure who the owner is, please contact jenkins-admin.
- Use SSH (**git://**) **endpoint** for Gitlab (HTTPS will not work)
- Use this node library example: https://jenkins.factset.com/job/examples/job/node-library/

  More example project configurations are here:
  https://jenkins.factset.com/job/examples/

## To use existing project as template

- **Name**: (enter project name)
- **Click Copy existing item**
- Enter **examples/node-library** in "Copy from"

  To copy from an existing app-qa-automation project, it would be like this:
  **app-qa-automation/**qa-protractor-common

  **Troubleshooting**: Currently, creating new project gets an exception, but you can ignore it. If it does not automatically create the project, click OK to continue.

## Project configuration

**Name**: (your project name)
**Description**: (your project description)
**Discard Old Builds**: Yes
  **Strategy**: Log Rotation
    **Max # of builds to keep**: 10
**Gitlab connection**: gitlab.factset.com
Check **Disable Build** (No new builds will be executed until the project is re-enabled.)
**JDK**: (Default)
Check **Restrict where project can be run**
**Label expression**: `jenkins_managed_linux`
**Source code management**: Git
**Repository URL**: git@gitlab.factset.com:example-group/example-library.git
**Credentials**: (use a service account)
**Branches to build**: origin/master
**Git executable**: git-default
**Repository browser**: gitlab
  **URL**: https://gitlab.factset.com/example-group/example-library/ (your Gitlab project root without the .git file extension)
  **Version**: 8.8 (only major.minor)
Build Triggers:
  **Build Periodically**: Yes
  **Schedule**: H/5 * * * *
Build Environment:

**Delete workspace before build starts**: Yes
**Abort the build if it is stuck**: Yes
**Time-out strategy**: Likely stuck
**Add timestamps to the console output**: Yes
Color ANSI Console Output
    **ANSI color map**: xterm
**Provide Node & npm bin/ folder to PATH**: Yes
**Installation**: node 4.x.x LTS

Build:
    **Execute Shell Command**:

```
# store temporary files within the workspace to avoid filling /tmp and to
avoid name collisions between instances of build tools
export TMPDIR="${WORKSPACE}/tmpdir"

# build project in production mode (https://gitlab.factset.com/developer-
services/developer-services/blob/master/workflows/builds.md#build-modes)
export GULP_ENV=production

# install project build dependencies
npm install

# build project, test build artifacts, and then build documentation.
npm run jenkins
```

**Post-build task script**:

```
if [ -x "$(npm bin)/semantic-release-gitlab" ]; then
  $(npm bin)/semantic-release-gitlab
else
  if [ -n "$(git tag -l --points-at HEAD)" ]; then
    npm publish
  fi
fi
```

**Note:** Jenkins build/post-build scripts are Linux bash shell scripts.

**Run script only if all previous steps were successful**: Yes
**Cobertura xml report pattern**: `coverage/**/cobertura-coverage.xml`
Reports:
    **HTML directory to archive**: `docs/`
    **Index page[s]**: index.html
    **Report title**: Source Code Documentation
E-mail Notification:
    **Recipients**: (team email address)
    Check **Send e-mail for every unstable build**
    Check **Send separate e-mails to individuals who broke the build**

- Jenkins Build will build Artifactory package if and only if a tag points to Git master branch HEAD. (That happens if you do `gulp release`)

## Publishing Package

Click Run Now in Jenkins job to manually publish package.

You must publish package right after `gulp release` and before next `git commit`, because it will only publish new releases that matches the latest git commit.

## Regression Test Suite

Regression test suite is compatible with Web application product version "**Live**". It can be used to run regression tests against prebuilds and QA builds. Regression test packages have npm dist-tag `regression`.

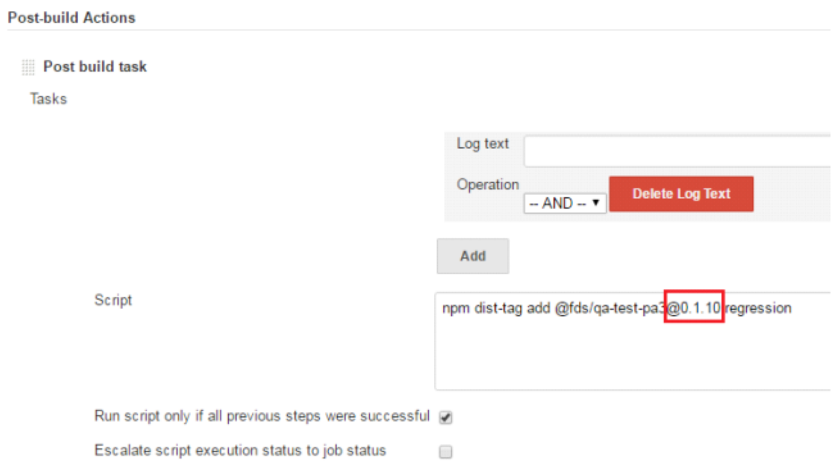
The Jenkins projects for building and publishing regression test packages have `_regression` suffix in the project name, for example, `qa-test-pa3`**`_regression`**. It is the same as production, except that instead of publishing a new version, it tags an existing version:

```
npm dist-tag add @fds/<pkg>@<version> <tag>
```

For example:

```
npm dist-tag add @fds/qa-test-pa3@170.1.3 regression
```

You need to go to the Jenkins project, click Configure, and change the @<version> part in "Post build task":


Post-build Actions

Post build task

Tasks

Log text

Operation -- AND --

Script

```
npm dist-tag add @fds/qa-test-pa3@0.1.10 regression
```

Run script only if all previous steps were successful ✓

Escalate script execution status to job status ☐

Click **Save**
Then click **Run Now** to execute it.

## List of dist-tags

To show versions associated with the dist-tags:

```
npm dist-tag ls @fds/qa-test-pa3
```

## Jenkins Regression Test-only Job Example

Example test-only Jenkins job (no project build): app-qa-automation/pa3-regression-test-example

**Example Jenkins script:**

```
json='
{
  "stage": "qa",
  "htmlReports": "staging",
  "serialNumber": 123456
}'
```

```
echo "$json" > test-config.json
npm install @fds/qa-test-pa3@regression
./node_modules/.bin/protractor node_modules/@fds/qa-test-
pa3/regression.conf.js
```

You must use a valid SerialNumber assigned to your service.

See https://gitlab.factset.com/app-qa-automation/qa-test-pa3 for usage information.

If the test fails, the Jenkins job will show Fail status, and optionally send email notification upon failure.

## Test configuration

The test configuration file is test-config.json in the current working directory on the command line. The minimum required properties are `Stage` and `SerialNumber`. If the file does not exist, the test will use these default values:

**Stage:** "Live",
**HTMLReports:** "Production"
**SerialNumber:** Joseph's serial number for easy of use for now, but the user should change it to his/her own

> Note: Regression test is compatible with "Live" Stage, so for demo purposes, that is the most stable, as sometime QA Stage might be incompatible.

The other configurations like SeleniumAddress (hub URL) and BrowserName and version are all standard protractor configurations that are configured in *conf.js, or as command line option --capabilities.browserName=chrmome, etc. But there is a currently a protractor bug that --capabilities.version for browser version does not work (but it works in conf.js.)

The regression.conf.js file in the PA3 test package (@fds/qa-test-pa3) is configured to be a ready-to-run regression test. The default configuration (proof-of-concept edition) is currently:

```
// Development hub
seleniumAddress: "http://tellusgridb04.pc.factset.com:4444/wd/hub",
capabilities: {
  'browserName': 'chrome',
  'version': '47',
},
specs: [
  'lib/e2e/document-options/asset-types/asset-add-remove.spec.js',
  'lib/e2e/document-options/asset-types/asset-type-search.spec.js'
],
```

A full set of *spec.js files will be added in the near future.

## Building Package

Click Build Now or schedule it in Configuration.

Please note that you need to run Build Now right after your gulp release but before other commits in that branch, else it won't find the git release tag to build.

`gulp release` to Gitlab repo is setup to trigger Jenkins build (via "Webhooks").

> **Note:** Multiple builds at the same time with the same repo might get errors. Please try to have

only one build session at a time for the same repo.

## User Administration

If you are admin of the Jenkins group/folder, you can add/modify user permissions here:
https://jenkins.factset.com/job/app-qa-automation/configure

## Reference

http://infonet.factset.com/view/Projects/DevServices/Jenkins

# Eslint auto fixing JSHint/JSCS lint errors

Tuesday, November 29, 2016 9:55 AM

## Introduction

Eslint auto fixes many lint errors reported by JSHint and JSCS, including,

- Line indentations
- Spaces inside curly braces
- Spaces after function names
- Space after `function` in anonymous function definition
  (And many more…)

Polyjuice can be used to convert JSHint and JSCS rules to ESLint rules.

### Setup

Notation: $ is command line prompt
(In your project root)

```
$ npm install -g eslint
$ npm install -g eslint-plugin-mocha
$ npm install -g polyjuice
```

Put this file in your project root and rename it *.eslintrc*:

eslintrc

Note: It was created from *.jshintrc* and *.jscsrc* by polyjuice like this (plus some customizations):

```
$ node_modules\.bin\polyjuice --jshint JSHINTRC --jscs JSCSRC > .eslintrc
```

## Usage

To fix lint errors:

```
$ eslint --fix src
```

The src is directory, and it will recurse into it.

## Rules Not Supported

- Blank line after block
  **Example:**

  ```
  // (blank line after the if-block)
  if (a == b) {
      // something
  }

  foo.somethingElse();
  ```

  Reference: http://stackoverflow.com/questions/36268195/enforcing-newlines-separating-code-blocks-in-eslint

## Rules not auto fixed

**Current not auto fixed, but could be enhance to:**

- Should have global 'use strict';
- Multiple var declarations in one statement, for example,
  ```
  var a,
      b,
      c;
  ```
  It should be,
  ```
  var a;
  var b;
  var c;
  ```
- No space allowed inside brackets, for example,
  // Incorrect:
  ```
  var stages = [ 'qa', 'devel', 'live' ];
  ```
  // Correct:
  ```
  var stages = ['qa', 'devel', 'live'];
  ```

**Probably should not be auto fixed:**

- Expected '===' and instead saw '=='.
- Variable is defined but never used.
- Variable is already defined
- 'timeout' is already defined.
- Error "Did you mean to return a conditional instead of an assignment?" for this:
  ```
  return this.a = b == c? b : c;
  ```

**Cannot auto fix:**

- Variable is not defined.
- Blocks are nested too deeply.
- This function's cyclomatic complexity is too high.

## Issues

- It inserts extra spaces in front of line comments:
  Bug: "standard --fix" alters indentation on comments #649
  Fix: #7618 (Pull Request)
- The following rules "space after comma" and "comma-dangling" generate space before closing curly, `{name: "value", }`:
  ```
      "comma-spacing": [
        2,
        {
          "before": false,
          "after": true
        }
      ],
      "comma-dangle": [
        2,
        "always-multiline"
      ],
  ```

which violates "no space inside curly braces" rule (for `` `{ name: value }` ``). So you will need to manually delete the space before the closing curly brace (`}`).

# TortoiseGit Installation and Usage

Thursday, February 12, 2015 9:15 AM

## Installation

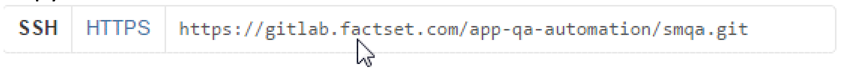
Download https://code.google.com/p/tortoisegit/wiki/Download
Run the installer to completion

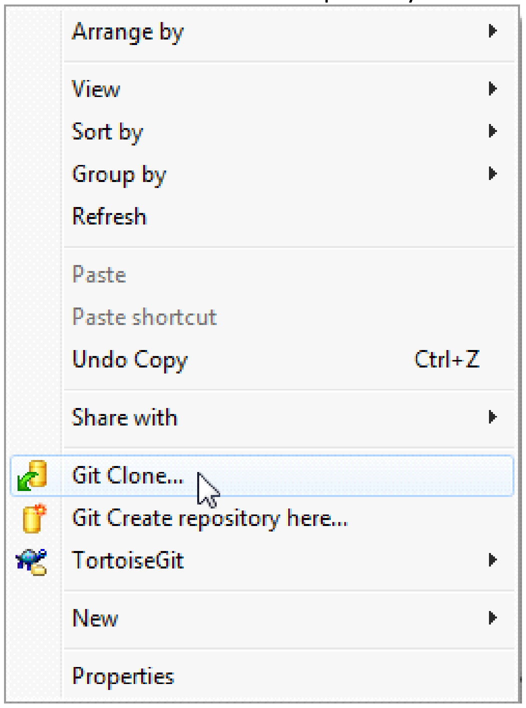
## Configuration

1. Select **Start >  All Programs > TortoiseGit > Settings**
2. Select **Git > Credential**
3. Select **"wincred - current Windows user"** for Credential Helper
4. Click **OK**

After this, TortoiseGit will prompt for password only once.

## Cloning a Repository

1. Go to the Gitlab project in browser, for example, https://gitlab.factset.com/app-qa-automation/smqa
2. Click HTTPS
3. Copy the HTTPS URL


https://gitlab.factset.com/app-qa-automation/smqa.git
4. Go to the parent of local projects in Windows Explorer
5. Right-click in any empty space
6. Click Git Clone to clone a remote repository:
   a. 

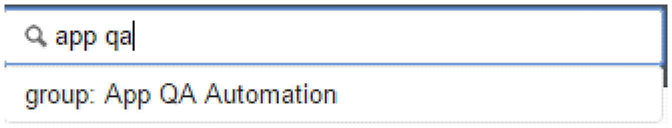
6. Paste the URL above to the URL text field
7. Click **OK**
8. Enter your Windows username (without domain) and Windows password when prompted

Documentation: https://code.google.com/p/tortoisegit/w/list

# Gitlab App QA Automation group

Thursday, February 12, 2015 7:53 AM

To see Gitlab group projects, search for App QA Automation:



One of the projects is:
[https://gitlab.factset.com/app-qa-automation/smqa](https://gitlab.factset.com/app-qa-automation/smqa)

# FactSet Gitlab Login

Tuesday, July 15, 2014 7:40 AM

As an employee, you should be automatically registered in Gitlab. To sign in:

1. Go to URL: https://gitlab.factset.com/users/sign_in
   (If you go to gitlab.factset.com, it will redirect to that.)
2. Click **LDAP** tab
3. Enter your Windows login ID without domain for **LDAP Login**
4. Enter your Windows login password for **Password**
5. Click **LDAP Sign in**

# Creating Gitlab project

Tuesday, July 15, 2014     8:13 AM

1. Login to Gitlab: Gitlab Login
2. Click **Dashboard** (on top) if you are not already there
3. Click **New project** (on right hand side):

4. Enter **Project name**
5. Enter a **Description**
6. Click **Private** for closed-source projects, or **Internal** for projects to be shared with the rest of the company, or **Public** for open source project
7. Click **Create project**

That creates an empty Gitlab project.

## Project Settings:

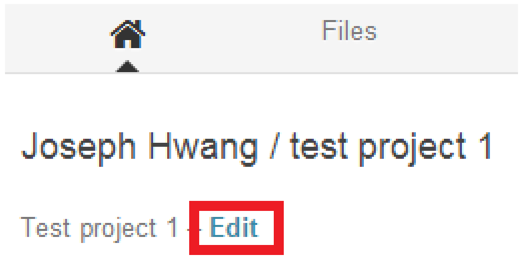
**Issue Tracker:** RPD
**Project name or id in issues tracker:** 6106

App QA Automation Product ID is 6106 in RPD.

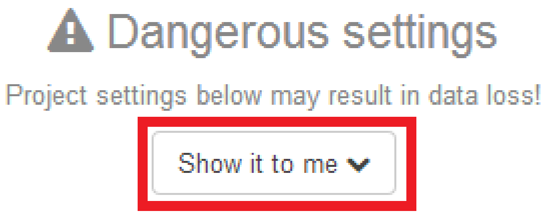
# Deleting Gitlab Project

Tuesday, July 15, 2014 11:05 AM

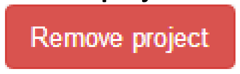
1. Open project home, for example, https://gitlab.factset.com/jhwang/test-project-1
2. Click **Edit**



3. Click "**Show it to me**" under "Dangerous settings"



4. Click **Remove project**



# Git Bash Installation

Thursday, February 12, 2015 7:42 AM

## Git Bash

This is a command line user interface.

1. Download Windows installer from http://git-scm.com/downloads
2. Run the .exe installer
3. Click **Next** (to accept default values)
4. For "Adjusting your PATH environment", select **"Use Git from the Windows Command Prompt"**
5. Click **Next**
6. For "Configuring the line ending conversions", select **"Checkout Windows-style, commit Unix-style line endings"**
7. Click **Next**
8. Click **Finish**

It will install command line Git, Git Gui, and Git Bash (UNIX-like command prompt.) The latter two will be under desktop **Start > All Programs > Git** menu.

# First time Git Bash client setup (for Git Bash only)

Tuesday, July 15, 2014 12:56 PM

You can do the following command in Git Bash.

On Windows systems, Git looks for the .gitconfig file in the $HOME directory (%USERPROFILE% in Windows' environment), which is C:\Documents and Settings\$USER or C:\Users\$USER for most people, depending on version ($USER is %USERNAME% in Wrcmdqindows' environment). It also still looks for /etc/gitconfig, although it's relative to the MSys root, which is wherever you decide to install Git on your Windows system when you run the installer.

## Your Identity

The first thing you should do when you install Git is to set your user name and e-mail address. This is important because every Git commit uses this information, and it's immutably baked into the commits you pass around:

```
$ git config --global user.name "John Doe"
$ git config --global user.email johndoe@example.com
```

Again, you need to do this only once if you pass the --global option, because then Git will always use that information for anything you do on that system. If you want to override this with a different name or e-mail address for specific projects, you can run the command without the --global option when you're in that project.

## Your Editor

Now that your identity is set up, you can configure the default text editor that will be used when Git needs you to type in a message. By default, Git uses your system's default editor, which is generally Vi or Vim. If you want to use a different text editor, such as Emacs, you can do the following:

```
$ git config --global core.editor emacs
```

## Your Diff Tool

Another useful option you may want to configure is the default diff tool to use to resolve merge conflicts. Say you want to use vimdiff:

```
$ git config --global merge.tool vimdiff
```

Git accepts kdiff3, tkdiff, meld, xxdiff, emerge, vimdiff, gvimdiff, ecmerge, and opendiff as valid merge tools. You can also set up a custom tool; see Chapter 7 for more information about doing that.

## Checking Your Settings

If you want to check your settings, you can use the git config --list command to list all the settings Git can find at that point:

```
$ git config --list
user.name=Scott Chacon
user.email=schacon@gmail.com
color.status=auto
color.branch=auto
color.interactive=auto
color.diff=auto
...
```

# Using existing Gitlab project on command line (Git Bash only)

Friday, July 11, 2014 1:21 PM

Note: You can use TortoiseGit or the command line.

1. **Install FactSet's SSL CA root certificate bundle:**

   ```
   $ git config --global http.sslcainfo c:/Users/<username>/Documents/certs/FactsetIntermediate.bundle.pem
   ```

2. **Clone the project (in Git Bash), for example,**

   ```
   $ git clone https://gitlab.factset.com/jhwang/protractor-helpers.git
   ```

3. If prompted, enter your Gitlab login ID (Windows login ID without Windows domain name) and password

   That will create a *<project-name>* sub-directory under the current directory.

4. Make changes to files
5. Commit local changes:

   ```
   $ git commit -am 'Update readme'
   [master 9d4f156] Update readme
    1 file changed, 1 insertion(+), 1 deletion(-)
   ```

   The -a option adds changes to the change list. The -m option sets comment.

6. Push changes (from local master branch) back to Gitlab:

   ```
   $ git push origin master
   ```

   Note: You may be prompted for Gitlab user name and password. Use your Windows login without the Windows domain name.

7. Another user can download your changes and merge into his/her local current branch like this:

   ```
   $ git pull origin
   remote: Counting objects: 7, done.
   remote: Compressing objects: 100% (4/4), done.
   remote: Total 4 (delta 0), reused 0 (delta 0)
   Unpacking objects: 100% (4/4), done.
   From gitlab.factset.com:jhwang/protractor-helpers
      bc71823..9d4f156  master     -> origin/master
   Updating bc71823..9d4f156
   Fast-forward
    dist/README.md | 2 +-
    1 file changed, 1 insertion(+), 1 deletion(-)
   ```

# Git Bash SSH Setup (optional)

Monday, July 14, 2014 9:18 PM

Note: This is for Git Bash and SSH only. You don't need this if you use TortoiseGit with https.

We strongly recommend using an SSH connection when interacting with GitHub. SSH keys are a way to identify trusted computers, without involving passwords. The steps below will walk you through generating an SSH key and then adding the public key to your GitHub account.

> **Tip:** We recommend that you regularly review your SSH keys list and revoke any that haven't been used in a while.

## Step 1: Check for SSH keys

First, we need to check for existing SSH keys on your computer. Open up your *Git Bash* and type:

```
ls -al ~/.ssh
    # Lists the files in your .ssh directory, if they exist
```

Check the directory listing to see if you have files named either id_rsa.pub or id_dsa.pub. If you don't have either of those files, go to step 2. Otherwise, skip to step 3.

> **Tip:** If you have a new installation of Mac OSX, you will not have a ~/.ssh directory. It will be created when you run the ssh-keygen command in step 2 below.

## Step 2: Generate a new SSH key

To generate a new SSH key, copy and paste the text below, making sure to substitute in your email address. The default settings are preferred, so when you're prompted to "Enter a file in which to save the key", just press Enter to continue.

```
ssh-keygen -t rsa -C "your_email@example.com"
# Creates a new ssh key, using the provided email as a label
# Generating public/private rsa key pair.
# Enter file in which to save the key (/c/Users/you/.ssh/id_rsa): [Press enter]
```

Next, you'll be asked to enter a passphrase.

> **Tip:** We strongly recommend a very good, secure passphrase. For more information, see Working with SSH key passphrases.

```
# Enter passphrase (empty for no passphrase): [Type a passphrase]
# Enter same passphrase again: [Type passphrase again]
```

Which should give you something like this:

```
# Your identification has been saved in /c/Users/you/.ssh/id_rsa.
# Your public key has been saved in /c/Users/you/.ssh/id_rsa.pub.
# The key fingerprint is:
# 01:34:f4:3b:ca:85:d6:17:a1:7d:f0:68:9d:f0:a2:db your_email@example.com
```

Then add your new key to the ssh-agent:

```
# start the ssh-agent in the background
eval "$(ssh-agent -s)"
```

```
# Agent pid 59566
ssh-add ~/.ssh/id_rsa
```

## Adding Your SSH Key to Gitlab

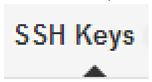
Run the following code in *Git Bash* to copy the key to your clipboard:

```
    clip < ~/.ssh/id_rsa.pub
```

It copies the contents of the id_rsa.pub file to your clipboard

Alternatively, using your favorite text editor, you can open the ~/.ssh/id_rsa.pub file and copy the contents of the file manually.

Now that you have the key copied, it's time to add it into GitLab:

1. Log in to Gitlab: Gitlab Login
2. On the toolbar, click **SSH Keys**:
   

3. Click **Add SSH Key**:

4. In the **Title** field, add a descriptive label for the new key. For example, if you're using a personal Mac, you might call this key "Personal MacBook Air".
5. Press **Ctrl+V** to paste your key into the "Key" field.
6. Click **Add key**:

### Reference

http://infonet.factset.com/view/Projects/DevServices/GitLab#SSH_Keys

# Uploading local project to Gitlab project (Git Bash)

Wednesday, July 09, 2014 7:27 AM

Need a Gitlab project first:

1. Create an empty Gitlab project:
   Creating Gitlab project
2. Get the Gitlab project's SSH URL handy for later, for example:
   `git@gitlab.factset.com:user1/project1.git`

Creating Local Git Repository and Upload to Gitlab:

1. Open *Git Bash*
2. **`mkdir myproject`**
3. **`cd myproject`**
4. Initialize as local Git repository:
   **`git init`**

   Note: That creates `.git` sub-directory under current directory.

5. Add any files or directories to local repo:
   ```
   git add README
   git add src
   ```

   Note: You can use wildcard like `file*.txt`.

6. To commit changes to local repo:
   **`git commit -m 'Add files'`**

7. Add reference to remote repository:
   `git remote add origin git@gitlab.factset.com:user1/project1.git`

8. Push pre-existing local commits to remote github:
   `git push -u origin master`

Now your project has migrated to Gitlab.

## Existing Git Repo?

```
1. cd existing_git_repo
2. git remote add origin git@gitlab.factset.com:user1/project.git
3. git push -u origin master
```

To show details about remote origin:

git remote --verbose

# Using folder-based "central" Git repository

Wednesday, April 06, 2016 11:27 AM

## Folder-based Git repository

Git does not have the concept of central server. A “client” could clone another repository. Git supports using a folder like a "central" repository. That folder could be a Windows Share, for example, `H:\Department\QualityAssurance\AppQA\GitRepo\`*`ProjectName`*.

### "Server"

To create a folder-based “central” repository, create that repository as if it were local:

```
cd H:\Department\QualityAssurance\AppQA\GitRepo\ProjectName
git init
git add <file>
git commit -m "commit message"
```

**Note:** You might want to configure gitignore before commit (https://git-scm.com/docs/gitignore).

### Client

The individual developer will clone the "central" repository:

```
git clone H:\path\to\central\repo_folder
```

That will create a local folder *repo_folder* in the current directory.

Then modify files and commit to local (developer) repository:

```
git commit -am "commit message"
```

To push it back to the "central" repository:

```
git push
```

That will push the local master branch by default.

Or explicitly push the master:

```
git push origin master
```

# Thief(Regression runs)

Tuesday, April 26, 2016 12:16 AM

Accounts:
1.accounts-overrides-fail(new issue )
2.acct-settings-pass
3.acct-restore-defaults(new issue )
4.disable-ofdb-cstm-issue
5.restore-defaults-fail( isue)
6.acct-settings2-issue

Audit:
1.audit-order-passed
2.audit-fixed-inc-passed
3.audit-security-passed
4.audit-pricing-passed
5.audit-grp-total-passed
6.audit-general-passed
7.audit-group-passed

Asset-types:
1.asset-general-passed

Trader-1:
1.trader-ca-exp-passed
2.trader-partial-duration-passed
3.trader-historical-attribution-passed
4.trader-long-short-passed
5.trader-same-sec-mult-port-passed

Trader-2:

1.trader-contribution-modeling-passed

Trader-3:

1. trader-liquidate-passed
2. trader-symbol-resolution-passed

Trader-4

1. trader-pre-post-trade-passed
2. trader-zero-price-passed

- Main-page-po

1. getWrenchIconFromReportWorkspace()
2. isReportCalculated()
3. getOptionFromWrenchMenuFromReportWorkspace()
4. getRowNameHavingSpecifiedAttribute()
5. getGroupingsHyperLink

- Audit.po

  1.isAuditMode()
  2. getOptionFromWrenchMenuFromAuditView()

- *STUTraderPage*

  *1.*setIdentifier()

- Docoptions-dates.po

  1.getDropDownItem()

- Doc-options-prices-advanced.po

6. getPortfolioSplitSourceDropdownButton()
7. getDropdownItem()

- *DocumentOptionsAssetTypeAddRemove*

  1.getOption()

- Tile-options.po.

  1.XpathTileOptionsTitle

- TileOptionsUniverse

  1.getExpandCompositeAssetsDropDown()
  2.getDropDownItem()

- Tile-options.columns.po
  1.getDropDownOptionFromDefinitionSection()

Fiddler:

1.audit-fixed-inc-------fail
2.audit-order-------fail
3.audit-general-----fail

http://is.factset.com/rpd/summary.aspx?messageid=23310759
Issue occurred in:
1.accounts-overrides
2.acct-restore-defaults
3Tile-options/dates/dates_blasting-deflt
4.Tile-options/grouping/grpg-edit-del

5.Tile-options/grouping/grpgs-accounts
6.toolbar/holding-bmark

# GIT FAQs

Tuesday, March 07, 2017 4:06 PM

**1) How to create a Jenkins Instance for new projects**
To Do: Joseph - Recording of a Demo (end to end) for a new project setup

**2) Updating the version in Package.json for dependent repos(page-objects) automatically?**
Ans: As of now, dependencies should be bumped first manually, then the final repo

**3) Is there an Automated process for updating the dependent repo version number?**
Ans: No
To Do: Any one of the following:

- Enhancement request for Developer Services to support updating the dependent repo version number or
- Framework - Fork and customize the developer services code – don't merge back to them
- Write new code in node js?

**4) Do we need a dedicated branch for CID/BVT?**
Ans : No as long as a specific conf.js is maintained

**5) Naming convention of versions in package.json should be?**
Ans: Tilde symbol - Regular tests, Page objects

**6) How do we decide if SCM should be GIT/Perforce?**
Ans: Based on the application and the engg code

**7) GIT Structure:**
Minor version(highlighted in yellow) indicates TellusDev or Tellus

Online:
The current structure in GIT has 6 branches: Devel.dev, Devel, QA.dev, QA, Live.dev and Live
With this, the weekly merge will become a tedious task as Merging of branches has to be done and 6 releases for the tests rep o and similarly 6 releases for Page objects repo for each application(12 releases).

Solution ->

- New branches with version numbers
- Major version should be the Online version

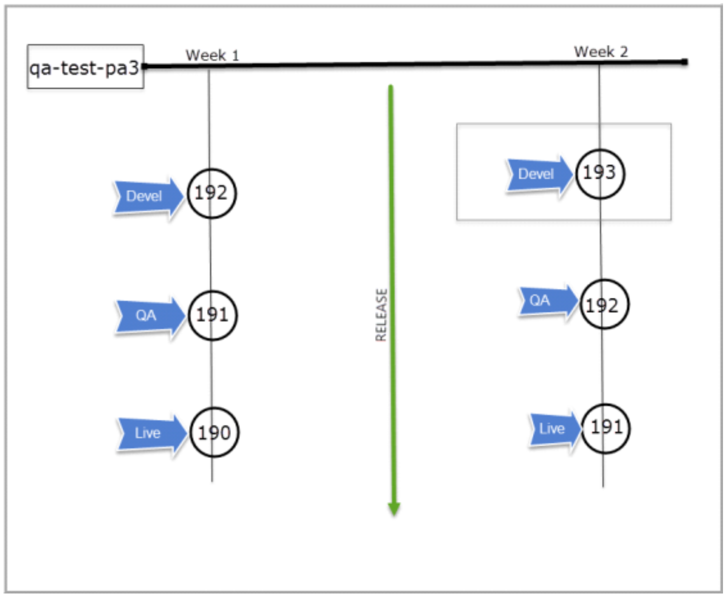
For online projects, each stage should be a branch in GIT and a version number in npm artifactory. For example, if the curren t Devel is 198, QA is 197 and Live is 196; we will have version196, version197 and version198 as branches in GIT and *v196.0.0, v197.0.0* and *v198.0.0* as versions in npm artifactory (we are giving version as prefix to branches as a standard). As the promotion happens every week and new version is available, we need to create a new branch in GIT and release it to npm artifactory to be able to test that version.

Note: when we release project to artifactory using "`gulp release`" it will automatically create a tag in git with the specified version number.

As per the below diagram a new branch and a tag will be created every week – of course you should create that.

Note: We want to have same versioning for both tests and page objects repos. So 191 version of tests should have 191 version of page objects as dependency in *package.json.*

Tellus will get only major version number from tracker so it runs a command like "`npm install @fds/qa-test-pa3@192`" if the current devel version is 192. With that command npm will download the latest major version available.



Pros:

- This way, you release a new branch only for Devel when there is a promotion
- No merging involved

Cons:

- Atlas will need to maintain Live system? No

Atlas:

- Needs only one version and latest patch can be picked
- Major version indicates Current year to help track the changes relative to a time frame

eg. Regular build -> 2017.0.0
    branchbuild -> 2017.0.0-branchbuildname

**Job entry page** will have two options for Atlas system **latest** and **prebuild**. Latest will pick the latest version of the module available in artifactory and prebuild will allow entering the version number or dist tag in the text filed and pick the particular version.

Pre-build:

**Prebuild testing** can be supported as prereleases. Please go through https://medium.com/@mbostock/prereleases-and-npm-e778fc5e2420 to understand the prereleases in npm.

Prereleases will also work as replacement for **testing in TellusDev**, which means, you can do a prerelease and test from Tellus and then release as actual version.

**8) What will the Test Case Location on JEP have?**
Ans:

**Online**: Devel, QA, Live and Prebuild/Branch [Tellus will translate the Test Case location on JEP based on Online Tracker]
**Atlas**: Latest, Prebuild/Branch

**Adhoc**: Devel, QA, Live, Prebuild, Latest

Note - Test Case location should support Perforce and GIT for a while, so the logic discussed about should be built into the Test Case population

**9) Source for downloading the scripts**
Ans: Artifactory

## Move to GIT support and CI Tasks - 03/08/17

Friday, March 10, 2017 2:36 AM

Update RPD:28884568 specs to support GIT structure – Update RPD - Krishnateja

Web hook modifications:
Modify the Web-hook to reflect versioning changes as described above – RPD TO BE FILED - Joseph
Can we enhance Web-hook to support passing a commit message? – Question for Joseph

Serial number for CID:
Framework
Get a dedicated pool for CI – RPD TO BE FILED - Vasu - RPD:30613439
Create an end-point to provide serial numbers and set caccess - RPD TO BE FILED - Vasu - RPD:30613559
It should return a response with S#, UserName, Password.
< Other type of S# request via our API-- stretch goal>
Create an end-point to release serial number - RPD TO BE FILED - Vasu - RPD:30614023
Setting the CACCESS, Provide the serial
Staf serial pool to be hosted on Staging

**Tellus Enhancement**
Store info(Serial number, FactSet .Net Username, FactSet .Net Password) in test-config.json(Should apply for Tellus jobs) - RPD TO BE FILED - Vasu, NOT for CI - **Separate RPD** RPD:30614154

Have a sample Test-Config.json and provide to functional team - Vasu - RPD:29920357

< High level time box Avanthi to update here >

CI
Enhance Jenkins/CI job config to do the following - RPD TO BE FILED - Rajul
- Request for serial number with Parameters (Caccess, type of pool)
- Should request to release the serial number at the end of the job
- Ability to update the TestConfig.js based on result from serial number API

Functional
Modify utilities in Selenium and Protractor to read the FactSet Username directly from test-config.json - RPD:29920357 - Eyme **< High Level time box 1 Day>**
Create utility to read from test-config.json for Selenium - RPD:29921572 - Eyme ( Selenium only )
**< High Level time box 1 Day >**

Communication process for changes during CI testing: TO DO -Eyme
Establish a workflow
Scripters for maintenance

## Move to GIT Functional Tasks - 05/12/17

- Ruby POC for GIT - 19th May
- Prepare Training – 22nd May
- Train automation team – 29th May

Move scripts to GIT:
- Fix Gulp lint errors
- Move scripts
- Update TTM
- Test on Tellus
- Update Cron jobs

### Better approach for Launching workstation for Protractor projects - 06/08/17

| Option | Implementation | Advantage | Disadvantage |
|---|---|---|---|
| **1) Continue to use the existing ruby utilities from the Selenium repo** | **• Maintain the Utilities in Ruby Common utilities gem itself and Protractor require these gems<br>• Add the gem as a Post install script which will take care of installation<br>• Drive the scripts using *RunProtractor.rb*** | **• Use existing code as such** | **Functional: Dependency on installing Ruby Utils gem<br>Framework: Maintain 2 different ways of parsing and calling files** |

| | | | |
|---|---|---|---|
| | • **Tellus continue to parse the ruby logs** | | |
| 2) Create a consolidated ruby utility and place it in the Protractor repo | • Lint setting ignores non .js files<br>• Drive the scripts using *RunProtractor.rb*<br>• Tellus continue to parse the ruby logs | • Use existing code by consolidating the existing ruby files.<br>• No dependency on installing Ruby Utils gem | Functional: Though changes to this utility is occasional, copy of the utility will be maintained in multiple places<br>Framework: Maintain 2 different ways of parsing and calling files |
| 3) Eliminate ruby, write an AutoIT script to launch Workstation, Tellus Parse the logs | • Drive it using *RunProtractor.js* and call conf.s based on the status of AutoIt run<br>• Tellus parse the AutoIT logs (test file?) since we need the details of the run or Tellus sends an email similar to the Job termination email with the error log | • Write new code to achieve this but we remove the dependency on other tools and instead all tools can use a common code<br>• Scripts get the control over initialization/startup | Functional: We still have a dependency on other ruby Utils for CEFMp anyway<br>Framework: Parse the AutoIt logs instead of ruby |
| 4) Eliminate ruby, write an AutoIT script to launch Workstation, Wrapper parses the logs | • Drive it using *RunProtractor.js* and call conf.s based on the status of AutoIT run<br>• *RunProtractor.js* parse the AutoIT log/txt file<br>• Tellus sends an email similar to the Job termination email with the error which is posted by *RunProtractor.js* to STDOUT | Functional:<br>• Write new code to achieve this but we remove the dependency on other tools and instead all tools can use a common code<br>• Scripts get the control over initialization/startup<br>• Framework: Tellus can parse just one log and set the status accordingly which is less confusing as a design | Functional:<br>• We still have a dependency on other ruby Utils for CEFMp anyway<br>• The wrapper *RunProtractor.js* has to be enhanced to parse the AutoIT log and throw error |

# Setting up Project on GIT and Running Job through GIT

Saturday, July 15, 2017 12:41 AM

| | |
|---|---|
| Subject | **RE: Setting up Project on GIT and Running Job through GIT** |
| From | Charuvaka Thallapelli |
| To | Durga Prasad Vemula; Komal Agrawal; Priyanka Kudikala; Ranvijay Kumar singh; Santhoshi Chepuru; Shireesha Gaddam; Sruthi Kura; Sudha Chakinala; Usha Sai Masabattula; V N Kiranmai Malladi; Vasavi Tallada |
| Cc | Eyme Ann Mathew; QAAutomation TLs |
| Sent | Friday, July 14, 2017 9:09 PM |

Hi All,

Hope this will help us in setting up new projects on GIT.

- **Setting up Project.**
  a. **Follow the Instructions in the Documentation prepared by Deekshith. -**

     Moving Automate...
  b. **Copy gulp.js file from any project that is already set up in GIT.**
  c. **Update Package.json file by using any project's package.json as reference.**
  d. **Include test>unit>test.spec.js file.**
  e. **Release Page objects as Node Modules (qa-page-objects-*).**
  f. **Include PageObjects node module as dependency in package.json of Tests (qa-test-*).**
  g. **Release Tests as a Node Module.**
  h. **Change TTM Settings.**
     i. **Test Path should be a full path includes test case name.**
- **Scheduling Runs from Tellus**
  a. **Use Prebuild as Test Case Stage.**
     i. **Let the Prebuild Input be empty if you want to pick the latest version.**
     ii. **Specify any version to test Specific version.**
- **How Tellus executes the Project.**
  a. **Before Downloading the archive, Tellus Checks for the Source Code Management Field from the Group ID.html file.**
     i. **If the SCM is set to GIT.**
        1. **It will copy CA_ACCESS.txt, serialNumber.txt files into Chunk Id folder.**
        2. **It will check for the Package Name Ex:(@fds/qa-test-bpm).**
        3. **And then It will install the package using npm install package-name@version(if Mentioned).**
        4. **Running npm install will download all the dependencies into node-modules folder inside chunk-id folder.**
        5. **If forgot to mention dependent files in package.json it will only download lib/ folder as it is included in package.json by default.**
        6. **Do not forget to include conf.js as a dependent file in package.json of Tests library.**
           "package.json", "README.md", "CHANGELOG.md", "CONTRIBUTING.md", "LICENSE", "lib" , "conf.js"
        7. **Do not forget to change require statements to require modules.**
           1) **Do a cntrl+shft+f , find all require statements and carefully replace with appropriate Modules.**

8. **Do not update conf.js instead maintain a dummy localConf.js file in order to test locally.**

- **Challenges with Testing Applications from GIT using SGE.**
  a. **Debugging.**
  b. **Root Cause Analysis.**
  c. **Finding session on which the Job is running.**
  d. **Project Archive size(when downloading archive to debug something) which includes Every node module instead we can limit to download only @fds folder which contains only project related modules.**

Regards,
Charvaka.

# Jenkins CI for Tellus App code deployment

Thursday, July 13, 2017 11:08 AM

**Workflow**

**Local Code Development in private branch( Commit to Laptop GIT)**
**>> Merge to Master Branch( Commit to GIT)**
**>> Jenkins pulls down and compile the code and deploy to Tellus Dev branch**
**>> Jenkins Releases to Production branch.**

Build Tool for building project which is dependent on Tech Stack
How to build various components with different technology base?
<https://www.ibm.com/developerworks/library/os-test-stafstax/index.html>,
<https://deployer.org/>
<https://github.com/git-ftp/git-ftp
https://wiki.jenkins.io/display/JENKINS/Publish+Over+FTP+Plugin
-- "may be" these can solve the purpose of copying the files over to server>

**Options**:
1. Build tools by technology base
2. Copy files to the SCM and the lets Jenkins take over ( STAF, PHP )

**Pre Event and Post Events**

Pre Events: -
Required Parameters to Trigger the Build
Deleting Old Builds
Have the ability to restore last working build or dependency

Post event:
Define and execute and deploy to target location/machine
Test are run and pass
Code Coverage
Email is sent out

**Assumption**: -
Code and Files should be same in Production and Dev
Separate curator for Production Deployment

**POC - Next step**

Define What all changes need to be done for getting for Jenkins?

# Git - Install CA root certificates

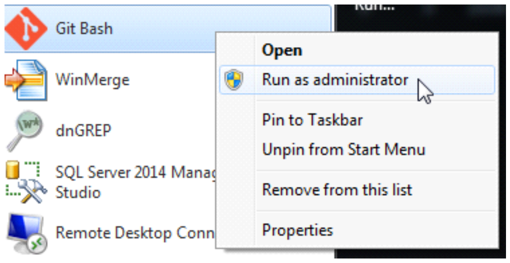
Friday, July 11, 2014 12:42 PM

You need to install FactSet's company SSL certificate before you can access Gitlab with git on command line.

The default certificate store (.crt) location is C:\Program Files (x86)\Git\bin\cur-ca-bundle.crt.

## Download and Install FactSet SSL Root Certificate:

**NOTE:** You need to run *Git Bash* as **administrator** for this:



```
$ curl http://infonet.factset.com/twiki/pub/Projects/DevServices/GitLab/factset-bundle.cer >> "C:\Program Files (x86)\Git\bin\curl-ca-bundle.crt"
```

That will append the certificate to the .crt file.

**-Or-**

1. Copy
   \\fds1.pc.factset.com\Apps\Git\Certificates\Certificates\ca-bundle.crt
   to a location on the C: drive.
2. In a cmd or Git Bash prompt, run:

```
git config --global http.sslCAInfo C:/Path/to/ca-bundle.crt
```

From <http://infonet.factset.com/view/Projects/DevServices/GitLab#HTTPS_SSL_Certificate>

## Reference

http://infonet.factset.com/view/Projects/DevServices/GitLab#HTTPS_SSL_Certificate

# QA CID Pipeline script development guide

Tuesday, July 18, 2017 9:53 AM

## Introduction

CID Pipeline scripts are a variant of groovy (with certain configuration and security restrictions.)

All CID Pipeline RPDs use the same Jenkins project (with conditional branching in code):
https://jenkins.factset.com/job/app-qa-automation/job/qa-cid-test/

It downloads scripts from Git repo:
https://gitlab.factset.com/app-qa-automation/qa-cid-test

The focus of the CID project is TestRunner.groovy script in the Git repo. Developer Services CID will download TestRunner.groovy from Git and call its RunTestsForApp() function when building FactSet products. It needs to handle any product passed to it. (Developer Services will access our Git repo but not our Jenkins project above.)

The Jenkinsfile script in Git repo is essentially a test runner for TestRuner.groovy.
The Jenkins project will automatically call the Jenkinsfile script of the Git repo.

- TestRunner.groovy:
  https://gitlab.factset.com/app-qa-automation/qa-cid-test/blob/master/TestRunner.groovy

## Test driving the Pipeline Script

- Go to Jenkins project, qa-cid-test:
  https://jenkins.factset.com/job/app-qa-automation/job/qa-cid-test/
- Click **Build Now** to run the Jenkins test

## Jenkins Configuration

Under Pipeline > Pipeline script from SCM,

**SCM:** Git
**Repository URL:** git@gitlab.factset.com:app-qa-automation/qa-cid-test.git
**Credentials:** git (SSH Key used for jenkins access on gitlab)
**Branches to build:** */master
**Git executable:** git-default
**Repository browser:** (Auto)
**Script Path:** Jenkinsfile
**Lightweight checkout:** Yes

## Protractor Environment Initialization

In Pipeline script:

```
  def node = tool name: 'node 6.x.x LTS', type:
'jenkins.plugins.nodejs.tools.NodeJSInstallation'
  env.PATH = "${node}/bin:${env.PATH}"
```

Then you can call node and npm like this:

```
  sh """
    node --version
    npm --version
```

```
"""
```

## Ruby Environment Setup

You can inject rbenv into your environment like so:

tool name: 'Linux rbenv with Ruby 2.2.2', type: 'com.cloudbees.jenkins.plugins.customtools.CustomTool'

If you click on a jenkins job that uses pipelines there is a helper to generates lines like the above. On the main job page there will be a link that says "Pipeline syntax". https://jenkins.factset.com/job/cid-staging/job/user/job/bbatha/job/cid-test-service/pipeline-syntax/

Once you have rbenv in your environment you'll need to execute the script lines you normally do to set up a ruby version, e.g.

```
`sh "rbenv shell 2.4"`
`sh "bundle exec rake"`
```

## TestRunner.groovy Description

- TestRunner.groovy runs the protractor testing like this:
    - Create test-config.json in the Jenkins project root in workspace
    - Run protractor:

      ```
      sh "./node_modules/.bin/protractor --params.path=${cfDir} --baseUrl=https://
      ${repoUrl}/ node_modules/@fds/qa-test-pru/cid.conf.js"
      ```

      where `cfDir` is directory of test-config.json.  and `repoUrl` is URL of product under test.

### Example TestRunner.groovy with Protractor Testing

```
import groovy.json.JsonSlurper

/**
 * Get version for stage from Tracker API.
 * @param appName
 * @param stage default "devel"
 * @param debug true for debug output
 * @return
 */
def getVersion(appName, stage="devel", debug=false) {
  /*
    Example response JSON:
    {"actions":[{"label_name":"online_2017_07_11_001","cluster":"fxdev1-
a","stage":"devel","label_minor":5,"status":1,"message":null,"stamp":"2017-07-11
18:41:10-04","user":null,"label_id":2736,"platform":"Fonix","id":141857,"type":"promo
te","patch":2,"label_version":203,"overwritten":0}],"last_stamp":"2017-07-11
18:41:10-04","status":1,"last_status":1}
   */
  def trackerUrl = "http://tracker.factset.com/build_status?stage=
${stage}&type=promote"
  def response = trackerUrl.toURL().text
  echo(response)
  def jsonSlurper = new JsonSlurper()
  def json = jsonSlurper.parseText(response)
  def actions = json.actions
  def rv = null
```

```
  if (actions.size() > 0) {
    def item = actions[0]
    rv = item.label_version
  }
  return rv
}

def RunTestsForApp(map) {
  // app-name string Name of the app
  // repo-url string URL to the git repo
  // commit string The commit being built
  // factset-json map The factset.json of the app
  // fdsav2-url string The FDSAv2 URL for the app. i.e;
<appname>.nprod.services.factset.com
  // e2e-label string The label generated for holding the deployment to be tested

  echo "RunTestForApp with "
  for (e in map) {
    echo "${e.key}: ${e.value}"
  }

  // Initialize environment
  def node = tool name: 'node 6.x.x LTS', type:
'jenkins.plugins.nodejs.tools.NodeJSInstallation'
  env.PATH = "${node}/bin:${env.PATH}"

  sh """
    node --version
    npm --version
  """

  def repoUrl = map["fdsav2-url"]
//  def repoUrl = "pru.staging-cauth.factset.com"
  def appName = map["app-name"]
//  def appName = "qa-test-pru"
  def versionSuffix = getVersion(appName)
  echo "Tracker version: ${versionSuffix}"
  if (versionSuffix != null) versionSuffix = "@^" + versionSuffix + ".0.0"
  def cfDir = '.'

  def json="""
    {
      "Stage": "Devel",
      "FDSAv2Stage": "Devel",
      "FDSAv2Overrides": "-1",
      "HTMLReports": "Staging",
      "PrebuildName": "",
      "BrowserType": "Chrome",
      "BrowserVersion": "54",
      "VMSUserID": "FDSQAR_C",
      "SerialNumber": "381970",
      "UserName": "automation.tc381970",
      "Password": "TellusAut0mation",
      "TestEnvironment": "SGE"
    }
    """

  writeFile file: "test-config.json", text: json

  try {
    sh """
      npm prune
      npm install protractor
      ./node_modules/.bin/protractor --version
      npm install @fds/${appName}${versionSuffix}
      ./node_modules/.bin/protractor --params.path=${cfDir} --baseUrl=https://
${repoUrl}/ node_modules/@fds/qa-test-pru/regression.conf.js
    """

  } catch (e) {
    emailext (
      to: "jhwang@factset.com",
      subject: "FAILED: Job '${env.JOB_NAME} [${env.BUILD_NUMBER}]'",
      mimeType: "text/html",
      body: """<p>FAILED: Job '${env.JOB_NAME} [${env.BUILD_NUMBER}]':</p>
         <p>Check console output at &QUOT;<a  href='${env.BUILD_URL}'>${env.JOB_NAME}
[${env.BUILD_NUMBER}]</a>&QUOT;</p>""",
      recipientProviders: [[$class: 'DevelopersRecipientProvider']]
    )
    throw e
  }

  echo "Tests complete!"
}

return this
```

## Pipeline Workspace

Test screenshots should be in the Workspace, which is in the project UI:
Ex: https://jenkins.factset.com/job/app-qa-automation/job/qa-cid-test/359/execution/node/3/ws/

* Select Jenkins Project
* Select the test run
* Select Pipeline Steps (one left)
* Select Allocate Node - Start (in tree on right)
* Select Workspace (on left)

## Documentation

- http://is.factset.com/rpd/summary.aspx?messageId=25994896&commentId=32090575
- https://jenkins.io/doc/book/pipeline/
- Example NodeJS environment setup:
  https://gitlab.factset.com/cid/cid-gitlab-webhook/blob/master/src/templates/Jenkinsfile.mustache#L5
- Example protractor setup:
  https://gitlab.factset.com/cid/cid-gitlab-webhook/blob/master/src/templates/Jenkinsfile.mustache#L38
- Pipeline examples:
- https://jenkins.io/doc/pipeline/examples/
- https://wiki.jenkins.io/display/JENKINS/NodeJS+Plugin

Older Documentation (some content might be outdated):

- https://readthedocs.factset.io/docs/cid/en/latest/
- https://gitlab.factset.com/cid-poc/cid-e2e-test
- https://jenkins.factset.com/job/cid/job/cid-e2e-test/

Wednesday, October 04, 2017 9:28 AM

AF-USA
CNUN-USA

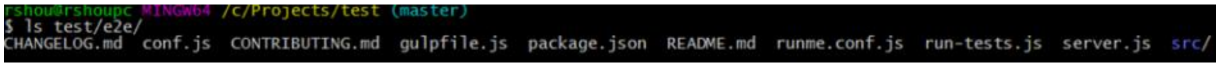
# git sparse checkout

Wednesday, November 22, 2017 2:03 PM

key: partial download import

## To download only e2e tests from PA3 git repo:

```
git init test
cd test
git remote -v
git remote add origin git@github.factset.com:rshou/pa3-frontend.git
git config core.sparsecheckout true
echo "test/e2e/*" >> .git/info/sparse-checkout
git pull --depth=1 origin feat/automation-test
```



```
rshou@rshoupc MINGW64 /c/Projects/test (master)
$ ls test/e2e/
CHANGELOG.md  conf.js  CONTRIBUTING.md  gulpfile.js  package.json  README.md  runme.conf.js  run-tests.js  server.js  src/
```

**Note:** You can put multiple folders or files in sparse-checkout file, one filespec pattern per line.

If you add another folder, `git pull` might not download it.  You need to run `git read-tree`:

```
git read-tree
```

## Reference

- From Rajesh Shou: https://briancoyner.github.io/2013/06/05/git-sparse-checkout.html
- https://git-scm.com/docs/git-read-tree

# QA NPM Jenkins Project Configuration Example

Wednesday, January 03, 2018 9:36 AM

Example Configuration of Jenkins qa-test-us Project for building npm artifacts.

## General

- Discard old build: Yes
    - Strategy: Log Rotation
    - Days to keep builds: (blank)
    - Max # of builds to keep: 10
- GitHub project:
    - Project url: https://github.factset.com/app-qa-automation/qa-test-us/
    - GitLab connection: gitlab.factset.com

## Office 365 Connector

- Restrict where this project can be run
    - Label Expression: jenkins_managed_linux

## Source Code Management

- Git
    - Repositories
        - Repository URL: git@github.factset.com:app-qa-automation/qa-test-us.git
        - Credentials: git (SSH Key used for jenkins access on github)
    - Branches to build
        - Branch Specifier (blank for 'any'): **
    - Git executable: git-default
    - Repository browser: (Auto)

## Build Triggers

- GitHub Pull Request Builder: Yes
    - GitHub API credentials: https://github.factset.com/api/v3: svc-jenkins
- Admin list: Pramod Devi
- Use github hooks for build triggering: Yes
- GitHub hook trigger for GITScm polling: Yes
- Poll SCM: No

## Build Environment

- Delete workspace before build starts: Yes
- Abort the build if it's stuck: Yes
    - Time-out strategy: Likely stuck
    - Time-out variable: (blank)
    - Time-out actions: Fail the build
- Add timestamps to the Console Output: Yes
- Color ANSI Console Output: No
- Provide Node & npm bin/ folder to PATH: Yes
    - Installation: node 6.x.x LTS

## Build

- Execute shell
    - Command:

```
# store temporary files within the workspace to avoid filling
/tmp and to avoid name collisions between instances of build
tools
export TMPDIR="${WORKSPACE}/tmpdir"

# build project in production mode
(https://gitlab.factset.com/developer-services/developer-
services/blob/master/workflows/builds.md#build-modes)
export GULP_ENV=development

branch_name="${GIT_BRANCH#origin/}"

git checkout "$branch_name"

# install project build dependencies
npm install

# build project, test build artifacts, and then build
documentation.
npm run jenkins
```

## Post-build Actions

Post build task

Tasks

Script

```
if [ -x "$(npm bin)/semantic-release-gitlab" ]; then
  $(npm bin)/semantic-release-gitlab
else
  if [ -n "$(git tag -l --points-at HEAD)" ]; then
    npm publish
  fi
fi

#Add dist-tag
branch_name="${GIT_BRANCH#origin/}"
if [ -n "$branch_name" -a "x$branch_name" != "xmaster" ];
then
    ver=$(node -e
"console.log(require('./package.json').version)")
    npm dist-tag add @fds/qa-test-us@$ver "$branch_name"
fi
```

Run script only if all previous steps were successful: Yes
Escalate script execution status to job status: No

publish Covertura Coverage Report

Cobertura xml report pattern: coverage/**/cobertura-coverage.xml

E-mail Notification

Recipients: pmuddalapura@factset.com pdevi@factset.com bindu.kaparthi@factset.com
Send e-mail for every unstable build: Yes
Send separate e-mails to individuals who broke the build: Yes

# qa-test-us test procedure

Friday, January 19, 2018 2:40 PM

Example testing with qa-test-us repo.

```
git clone git@github.factset.com:app-qa-automation/qa-test-us.git
git checkout version223
```

You can also download the test script from github.com:

- Select the branch
- Click Download

```
cd qa-test-us
npm install
```

### Compile the code:

```
gulp build
```

## Acquiring Serial Number and Setting FDSAv2 Overrides

Before running the test, you need to get serial number and set FDSAv2 overrides via Serial Number API:

```
curl -sk -XPOST -d '{"RequestJson":{"serialnumber":"686363","vmsusername":"FDSQAR_C",
"fdsav2stage":"qa","fdsav2overrides":"review_universal-screening_moc-breakautomation_2426"}}' -
H "Content-Type:application/json" -H "Accept:application/json"
http://tellusstgb01:8080/tellusci/ci/request
```

Note: This example uses 686363, matching that of test-config.json below, but usually one should use `"serialnumber":"random"` , in order to use *any* number. And you need to put the acquired serial number in test-config.json below.

## Creating Test Configuration

You need to create test-config.json in the current working directory. Use the serial number acquired above.

### test-config.json:

```
{
  "Stage": "qa",
  "FDSAv2Stage": "qa",
  "FDSAv2Overrides": "review_universal-screening_moc-breakautomation_2426",
  "HTMLReports": "Staging",
  "PrebuildName": "",
  "SGBrowserName": "chrome",
  "SGBrowserVersion": "61",
  "VMSUserID": "FDSQAR_C",
  "SerialNumber": "686363",
  "UserName": "automation.tc686363",
  "Password": "TellusAut0mation",
  "TestEnvironment": "SGE",
  "SeleniumGridHub": "http://tellusdevb03.pc.factset.com:4444/wd/hub"
}
```

Change SeleniumGridHub to your own hub URL.

### To run test:

```
node_modules/.bin/protractor cid.conf.js
```

## Releasing Serial Number

After testing, you need to release the serial number:

```
curl -sk -XPOST -d '{"RequestJson":{"serialnumber":686363}' -H "Content-
Type:application/json" -H "Accept:application/json"
http://tellusstgb01:8080/tellusci/ci/release
```

# git delete branch

Tuesday, March 13, 2018 2:30 PM

To delete local branch:

```
git branch -d my_branch
```

To delete remote branch:

```
git push origin --delete <branch_name>
```