http://not4j.com/jenkins-how-to-start-jenkins-build-using-commit-id-of-last-successful-build-on-test-enviroment/

[Jenkins: How to start Jenkins build using commit id of last successful build on test enviroment?](https://twitter.com/intent/tweet?text=Jenkins%3A%20How%20to%20start%20Jenkins%20build%20using%20commit%20id%20of%20last%20successful%20build%20on%20test%20enviroment%3F%20%C2%BB&hashtags=jenkins,CI,groovy,continuous-integration&url=http://not4j.com/jenkins-how-to-start-jenkins-build-using-commit-id-of-last-successful-build-on-test-enviroment/" \o "Tweet 'Jenkins: How to start Jenkins build using commit id of last successful build on test enviroment?')

Last week I faced with a quite not trivial case in CI practice. There is production build running once a day by schedule and it is using master branch as a source. The main challenge here that the development team is not using branches in everyday practice (why? it is quite another topic, but at the moment we stay with one branch).

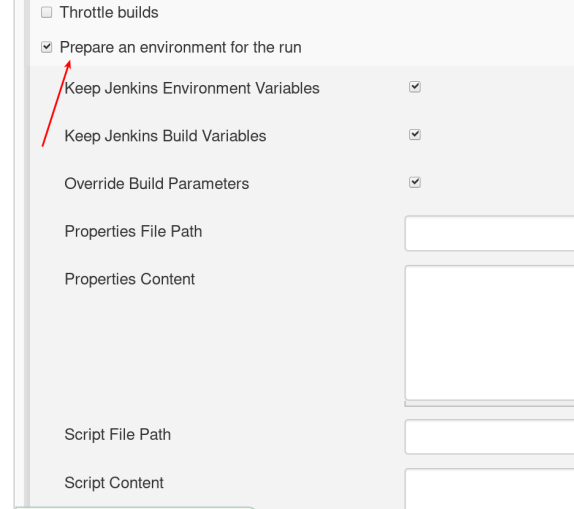
Such setup was working fine until dev team size hits 4 engineers. It becomes often case, that Jenkins build start while in the master branch we have some unfinished task that is in progress. It becomes obvious that we need use for the production build only the commit that produces the last successfull build on stage environment. Sounds good and should quite easy to implement, but...

Jenkins is great in case you have a pipeline of builds (one build trigger another one right after successful completion). However, it is not so easy to push a scheduled job to check the state of another job and obtain commit id in pre-build actions. At least, I do not find anything out of the box.

Let's try to create it on our own. First of all, we need to obtain the last successfull build with all additional info. Checking of [Jenkins api](https://wiki.jenkins.io/display/JENKINS/Remote+access+API) give us a link to do it: http://<jenkins\_url>/job/<job\_name>/lastSuccessfulBuild/api/json.

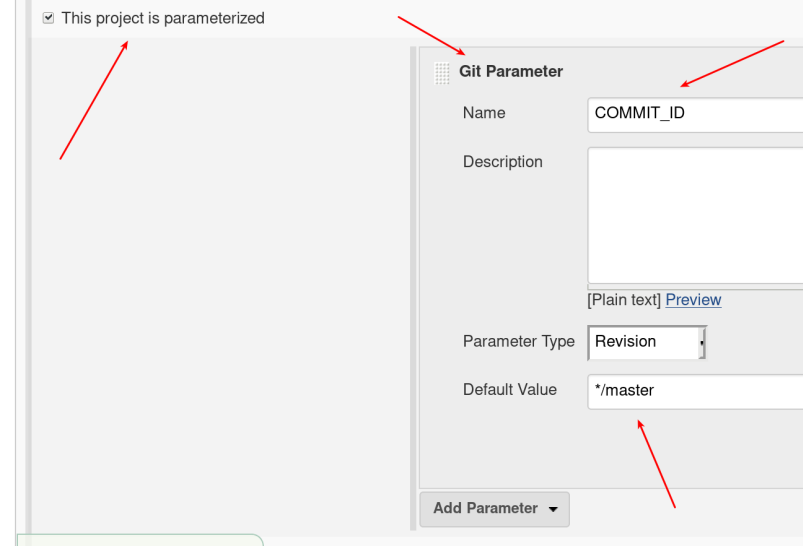
It will produce a huge JSON output, but the most intersing part is here: "lastBuiltRevision":{"SHA1":"<commit\_hash>"}. We could reduce JSON size, in case pass exact names of fields and depth of search over JSON: http://<jenkins\_url>/job/<job\_name>/lastSuccessfulBuild/api/json?tree=actions[lastBuiltRevision[SHA1]]&depth=3. OK, commit hash can be onbtained.

Now, we need to trigger this API call during pre-build actions, parse commit hash and use it for build start. It can be done with [Jenkins EnvInject Plugin](https://wiki.jenkins.io/display/JENKINS/EnvInject+Plugin). Setup it and in build parameter you should enable the option Prepare an environment for the run.

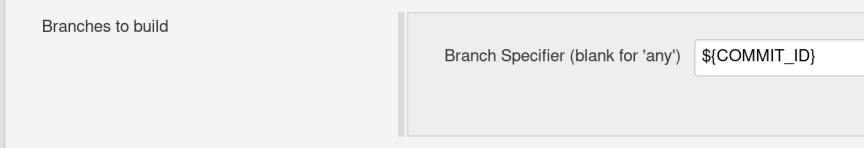


Do not forget to enable option Override Build Parameters. Ok, now we have an opportunity to override environment variables before build starts and can use such purpose: hardcode values, script (not sure what kind of script can be used, because I find out next option), groovy script. Groovy is a perfect language for me for such kind of automatization, so I start using this option. What else do we need? Ah... We can override env variable, but what variable to override? Let's create such one.

In build parameters you should enable the option This project is parameterized. Add Git parameter section. Now name parameter (I have used COMMIT\_ID) and set the default value of the parameter (in my case, it was a reference to master's head).



Let's inject a new environment variable. It can be done in the section Source Code Management, put reference to env variable in a field Branch Specifier (blank for 'any').



${COMMIT\_ID}

And the last step: perform API call, parse the hash and put it in env variable COMMIT\_ID. Let's return to Jenkins EnvInject Plugin and find section Groovy script. Put in the field the script:

def post = new URL("http://<jenkins\_url>/job/<job-name\_to\_check>/lastSuccessfulBuild/api/json?tree=actions[lastBuiltRevision[SHA1]]&depth=3").openConnection()

String user = "<user>"

String pass = "<API token>"

String authStr = user +":"+ pass

String encoding = authStr.getBytes("utf-8").encodeBase64().toString()

post.setRequestMethod("POST")

post.setDoOutput(true)

post.setRequestProperty("Authorization", "Basic " + encoding)

def postRC = post.getResponseCode()

if(postRC.equals(200)) {

def result = post.getInputStream().getText()

def shaBegin = result.indexOf('SHA1":"') + 7

def shaEnd = result.indexOf('"', shaBegin)

println(result.substring(shaBegin, shaEnd))

return ["COMMIT\_ID": result.substring(shaBegin, shaEnd)]

}

The script is pretty straightforward. You need to pass you username and API token (click on your username in Jenkins at the top -> Configure -> Show Legacy API Token). Next, we just parse hash from result string and return map with parameter name COMMIT\_ID and hash value.