

Jenkins Pipeline

Pipeline As Code

Topics

- Introduction
- Pipeline Basic
- Variables
- parameters
- option sets
- trigger builds
- schedule jobs
- parallel
- post jobs
- tools
- conditional and loop statements
- other examples
- sample maven build
- archive artifacts and finger prints
- uses of credentials option
- checkouts_and_execute_steps
- input section
- scm git
- when
- sample local deployment

Introduction

Pipeline script

- Another way of job configuration with help of code

Advantages:

- Can divide the jobs into parts (build /test /deploy/..) & each part can run in each agent.
- Parallel execution of stages are easy to configure so that we can save time
- Each stage can execute with different version of JDK/MVN versions
- Can retrigger from failed stage
- visualize the build flow
- Build can hold for user input (specific user can enter, explain LDAP concept)
- Version control, code review
- pause, restart the build
- In multibranch pipeline scripts will automatically create in subbranches

Types of Pipeline

- Declarative
- scripted

Difference between Declarative and scripted

- Declarative pipeline is a recent addition.
- More simplified and opinionated syntax when compared to scripted **Declarative syntax**

```
pipeline { agent
any stages {
stage('Build') {
steps {
//
}
}
stage('Test') {
steps {
//
}
}
stage('Deploy') {
steps {
//
}
}
}
```

Scripted Syntax

```
node {
stage('Build') {
//
}
stage('Test') {
//
}
stage('Deploy') {
//
}
}
```

PIPELINE BASIC

- ❖ Steps, Stage, Stages, agent sections
- ❖ Comments
- ❖ Pipeline Syntax
- ❖ Hello World
- ❖ Batch commands

Steps

- We need to write step inside the stage directive
- steps contains (command/scripts) that we used in the build
- One steps directive should be there in stage directive

Stage

- Define particular stage (build/test/deploy/..) of our job
- atleast one stage has to be there
- name will be display on the jenkins dashboard

stages

- contains sequence of stages
- atleast one stage has to be there

Agent

- where (master/slave/container..)we need to run our pipeline script

Stage colors

- White (stage is not executed)
- Green (stage is success)
- Blue lines (stage is executing)
- Redlines or red line (stage is failed)
- Red (few stage success, any one is failed, few remain success stage will show red) **Comments**
Single line comment : //

Multiline comment: /*

Declarative Pipeline Syntax

```

pipeline{ //pipeline declaration
  agent <option> //where
  stages{ //stages declaration
    stage("StageName"){ //Stage type : build/test/deploy
      steps{ /* Actual execution starts from here, like
               commands execution, scripts execution etc */
        echo "Hello World" //echo is the print statement
      } // Closed curly brace for steps section
    } //Stage
  } //stages
} // pipeline

```

Simple Hello world pipeline:

```

pipeline{
  agent any
  stages{
    stage('Hello_world'){
      steps{
        echo 'Hello world'
      }
    }
  }
}

```

O/P

Stage View



Batch commands

```

pipeline{  agent
any  stages{
stage('batch'){
steps{

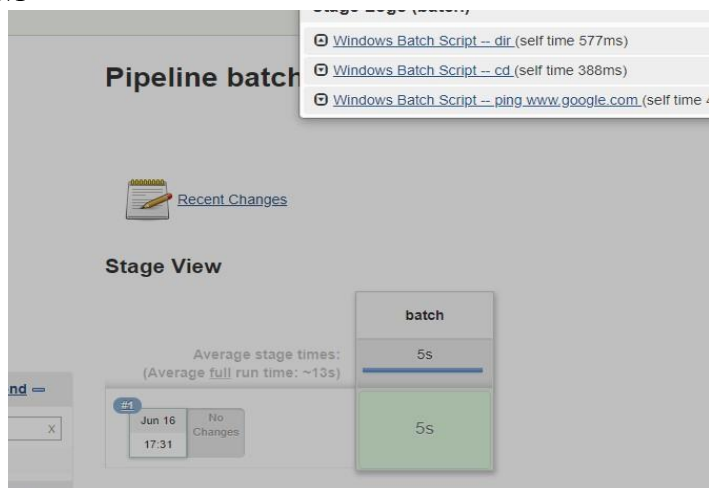
```

```

bat "dir"
bat "cd"
        bat "ping www.google.com"
    }
}
}
}
}

```

O/P



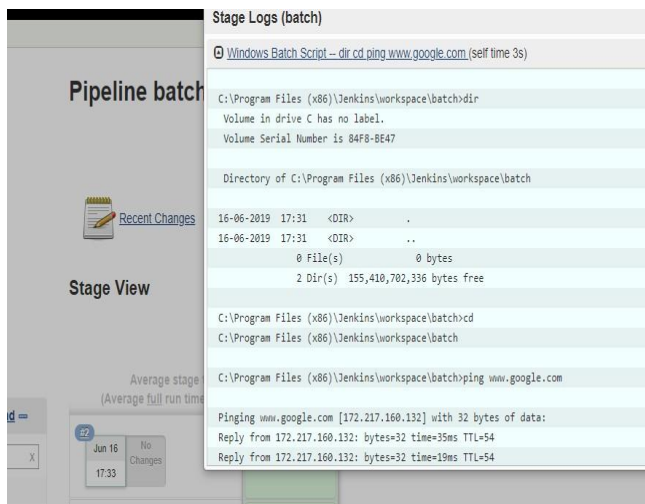
Multiline bat command

```

pipeline{
    agent
    any stages{
        stage('batch'){
            steps{
                bat """
                dir
                    cd
                    ping www.google.com
                """
            }
        }
    }
}

```

O/P



VARIABLES

Variables

What is the use of Variables

Pre defined Variables

User defined Variables

Scope of Variables

User defined VS Pre defined variables

Params, Difference between Single and Double quotes

Read variables from JSON file

Concatenation

What is variable?

Variable is used to store the value.

<variable name> = <variable value>

Types

- Predefined variable
- User variable

Predefined:

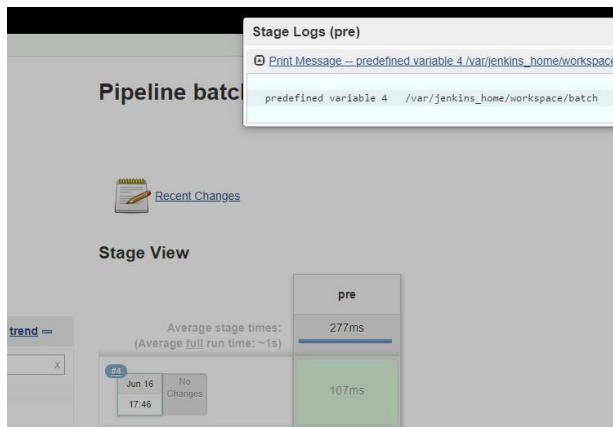
<http://localhost:8080/env-vars.html>

Predefined

```

pipeline{
  agent any
  stages{
    stage('pre'){
      steps{
        echo " predefined variable $BUILD_NUMBER  $WORKSPACE "
      }
    }
  }
}

```

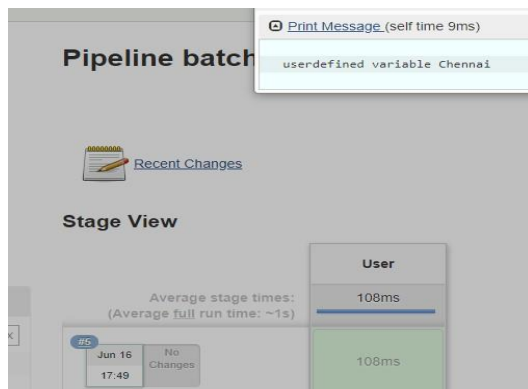


Userdefined : variable we can define in rootlevel or stage level

```

pipeline{
  agent any
  environment{
    MYHOME="Chennai"
  }
  stages{
    stage('User'){
      steps{
        echo " userdefined variable $MYHOME  "
      }
    }
  }
}

```



User defined variables in

- Global level
- stage level
- script level **Global level**

```
pipeline{
  agent any
  environment{
    MYHOME="Chennai"
  } stages{
  stage('User'){
    steps{
      echo " userdefined variable $MYHOME  "
    }
  }
}
}
```

Stage level

```
pipeline{
  agent any
  stages{

    stage('User'){
      environment{
        MYHOME="Chennai"
      }
    }
    steps{
      echo " userdefined variable $MYHOME  "
    }
  }
}
```

Script level

```
pipeline{
  agent any
  stages{

    stage('User'){

      steps{
        script{
          MYHOME="Chennai"
        }
      }
      echo " userdefined variable $MYHOME  "
    }
  }
}
```



```

}

pipeline{
  agent any
  stages{

    stage('User'){

      steps{
      script{
        MYHOME="Chennai"
        echo " userdefined variable $MYHOME  "
      }
    }
  }
}

```

Scope of the Variables : priority order first (script),second(stage),third(global or root)

if you defined the same variable in global ,stage and , it will pick up stage.

```

pipeline{
  agent any
  environment{
    MYHOME="Chennai"
  }
  stages{

    stage('User'){
  environment{
    MYHOME="thiruverkadu"
  }
    }
  }
}

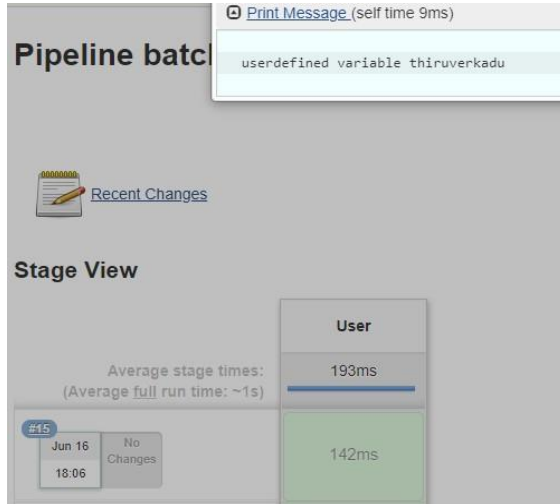
```

```

    }
  steps{
    echo " userdefined variable $MYHOME  "
  }
}
}
}
}

```

O/P



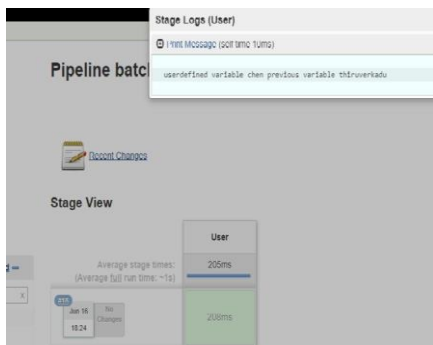
Predefined vs user defined values:

if you defined diff values in variable , we can call above stage variable by `${env.variablename}`

```

pipeline{
  agent any
  environment{
    MYHOME="Chennai"
  }
  stages{
    stage('User'){
      environment{
        MYHOME="thiruverkadu"
      }
      steps{
        script{
          MYHOME="chen"
        }
        echo " userdefined variable $MYHOME previous variable ${env.MYHOME}  "
      }
    }
  }
}

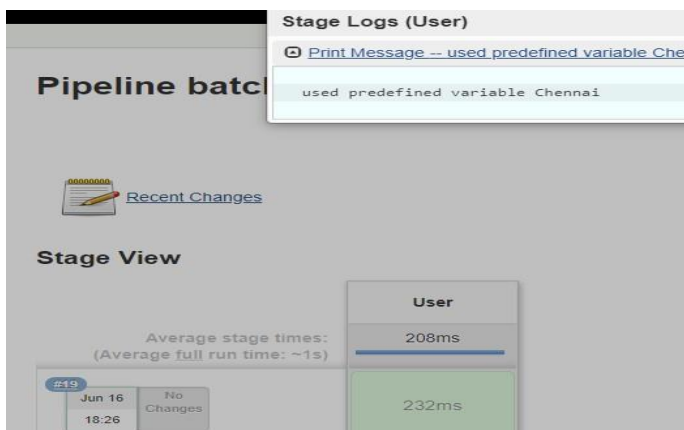
```



Eventhough it predefined variable if we change for custom, priority for user defined

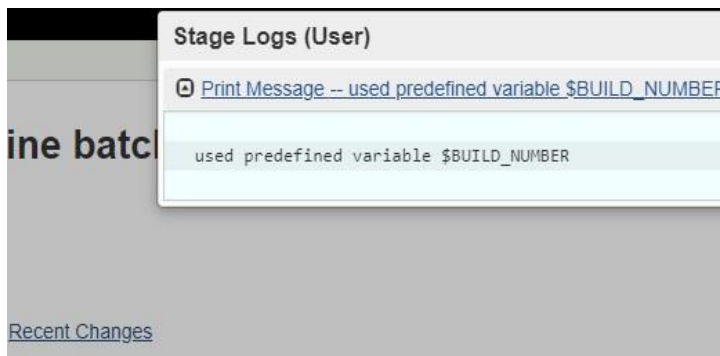
```
pipeline{
agent any
environment{
    BUILD_NUMBER="Chennai"
}
stages{

    stage('User'){
environment{
    MYHOME="thiruverkadu"
}
steps{
script{
    MYHOME="chen"
}
echo " used predefined variable $BUILD_NUMBER  "
}
}
}
}
```

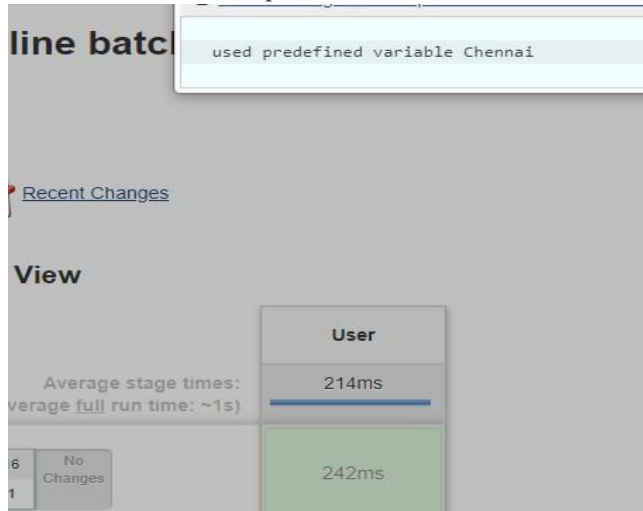


Diff B/W Single and Double quotes if we

defined in single quote it will take as string



If we defined in double quotes, it will take as variable name

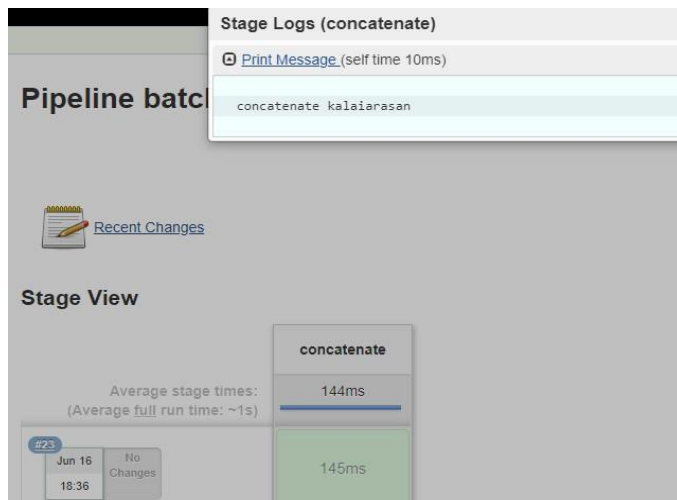


Concatenate process of combining two or more string by '+' operator in jenkins

```

pipeline{
  agent any
  environment{
    name1="kalai"
    name2='arasan'
  }
  stages{
    stage('concatenate'){
      steps{
        script{
          Name= name1 + name2
          echo " concatenate $Name"
        }
      }
    }
  }
}

```



PARAMETERS

Parameters :

Are used to pass the data dynamically

- ❖ String
- ❖ Text
- ❖ Boolean
- ❖ Choice
- ❖ Password
- ❖ File
- ❖ Dry Run

Syntax:

\$VARIABLENAME and params. VARIABLENAME is same. pipeline{ agent any parameters { string(name: 'DEPLOY_ENV', defaultValue: 'staging', description: ") text(name: 'DEPLOY_TEXT', defaultValue: 'One\nTwo\nThree\n', description: ") booleanParam(name: 'TOGGLE', defaultValue: true, description: 'Toggle this value')

choice(name: 'CHOICE', choices: ['One', 'Two', 'Three'], description: 'Pick something')

file(name: 'FILE', description: 'Some file to upload') password(name: 'PASSWORD', defaultValue: 'SECRET', description: 'A secret password') }

```
stages{
stage('string'){
    steps{
        echo " string $DEPLOY_ENV"
```

```

    }
  }
  stage('text'){

  steps{
    echo " text $DEPLOY_TEXT"
  }
}

stage('booleanParam'){

steps{

    script{
      if(TOGGLE){
        echo " now execute, booleann is true"
      }else{
        echo " Dont execute, boolean is true"
      }
    }
  }
}

stage('choice'){

steps{

    script{ if(DEPLOY_ENV=='staging'){
      echo " choice $CHOICE"
    }
  }
}

stage('file'){

steps{
  echo " file $FILE"
}
}

stage('password'){

steps{
  echo " password $PASSWORD"
}
}
}
}

```

O/P

Stage View

	string	text	booleanParam	choice	file	password
Average stage times: (Average full run time: ~2s)	97ms	92ms	293ms	91ms	120ms	83ms
#15 Jun 16 19:36 No Changes	177ms	56ms	131ms	120ms	67ms	73ms

Dryrun

Dryrun is mainly used for first time of parameter build, before getting build with parameter.

```

2 agent any
3 parameters {
4     choice(name: 'DryRun', choices: "Yes\nNo", description: "Do you need dry run?")
5     string(name: 'PERSON', defaultValue: 'Mr Jenkins', description: 'Who should I say hel')
6     text(name: 'BIOGRAPHY', defaultValue: '', description: 'Enter some information about t')
7 }
8 stages {
9     stage("parameterizing") {
10        steps {
11            script {
12                if ("${params.DryRun}" == "Yes") {
13                    currentBuild.result = 'ABORTED'
14                    error('DRY RUN COMPLETED. JOB PARAMETERIZED.')
15                }
16            }
17            echo "${PERSON}"
18        }
19    }
20 }
  
```

OPTION SET

❖ Options

- ❖ retry
- ❖ buildDiscarder
- ❖ disableconcurrentbuild
- ❖ timeout: timestamps

Options stage level or pipe level

- Retry: before failing the job, will run the job again to specified times
 - buildDiscarder : used to delete old build logs in number or days
 - disableConcurrentBuilds: used to disable concurrent build
 - Timeout: Time set for particular build
 - timestamp: will add the time to the build process
- Retry Stage based pipeline** { agent any
- ```

stages {
 stage('Deploy') {

```

```

 options { retry(3)
 timeout(time: 5, unit: 'SECONDS')
 }
 steps {
sh 'echo hello' sleep(10)

 }
 }
}

```

### Retry: step based

```

pipeline { agent any
stages {
stage('Deploy') {
steps { retry(3)
{ sh 'echo
hello' }

 }
 }
}
}

```

### Retry: global based

```

pipeline {
agent any
options{
retry(3)
}
 stages {
stage('Deploy') {
steps {
sh 'echo hello'

 }
 }
}
}

```

if any error or timeout it will execute 3 times



## Stage View



## buildDiscarder

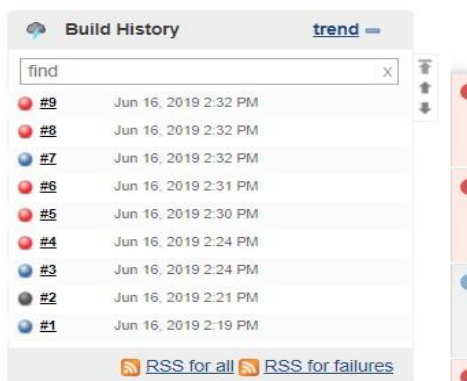
- numbers: options { buildDiscarder(logRotator(numToKeepStr: '5')) }
- days: options { buildDiscarder(logRotator(daysToKeepStr: '7')) }

```
pipeline {
 agent any
 options {
 buildDiscarder(logRotator(numToKeepStr: '5'))
 }
}
```

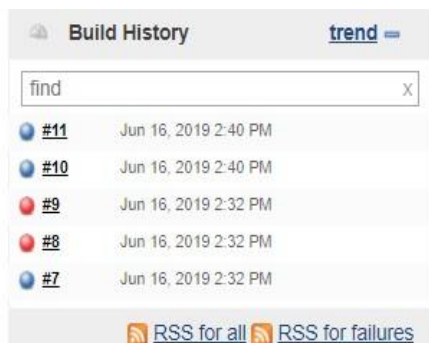
```
 stages {
 stage('Deploy') {
 steps {
 sh 'echo hello'
 }
 }
 }
}
```

```
 }
}
} before : buildDiscarder
```

execution



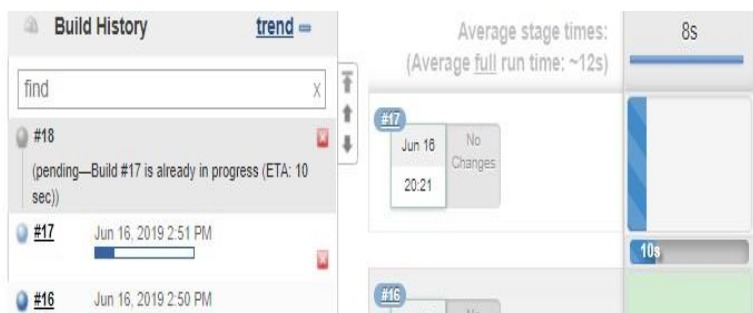
After : buildDiscarder execution



## disableConcurrentBuilds

if execute the build if it takes time to complete again parallel , we trigger b4 complete the previous build, again build get start to execute, due to this job will get conflicts with nodes.

```
pipeline {
 agent any
 options {
 buildDiscarder(logRotator(numToKeepStr: '5'))
 disableConcurrentBuilds()
 }
 stages {
 stage('Deploy') {
 steps {
 sh 'echo hello'
 sleep(10)
 }
 }
 }
}
```



## Timeout:

```
timeout(time: 30, unit: 'MINUTES')
```

```
timeout(time: 30, unit: 'SECONDS')
```

```
timeout(time: 30, unit: 'HOURS') Syntax
```

```
pipeline {
 agent any
 options {
 buildDiscarder(logRotator(numToKeepStr:
 '5')) disableConcurrentBuilds() timeout(time:
 5, unit: 'SECONDS')
 }
 stages {
 stage('Deploy') {
 steps {
 sh 'echo hello' sleep(10)
 }
 }
 }
}
```

```
}
```

its aborted after the timelimit



#### Console Output

```
Started by user kalai
Running in Durability level: MAX_SURVIVABILITY
[Pipeline] Start of Pipeline
[Pipeline] node
Running on Jenkins in /var/jenkins_home/workspace/retry
[Pipeline] {
[Pipeline] timeout
Timeout set to expire in 5 sec
[Pipeline] {
[Pipeline] stage
[Pipeline] { (Deploy)
[Pipeline] sh
+ echo hello
hello
[Pipeline] sleep
Sleeping for 10 sec
Cancelling nested steps due to timeout
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // timeout
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Timeout has been exceeded
Finished: ABORTED
```

#### Timeout Stage based:

```
pipeline {
 agent any

 stages {
 stage('Deploy') {
 options {
 retry(3)
 }
 timeout(time: 5, unit: 'SECONDS')

 steps {
 sh 'echo hello'

 sleep(10)
 }
 }
 }
}
```

#### Timestamp:

```
pipeline {
 agent any
 options {
 buildDiscarder(logRotator(numToKeepStr:
 '5')) disableConcurrentBuilds() timestamps()
 }

 stages {
 stage('Deploy') {
 steps {
 sh 'echo hello' sleep(2)
 sh 'echo hi'
 sleep(2)
 sh 'echo how'
 }
 }
 }
}
```

```
}
}
```

With timestamp



## Console Output

```
Started by user kalai
Running in Durability level: MAX_SURVIVABILITY
[Pipeline] Start of Pipeline
[Pipeline] node
Running on Jenkins in /var/jenkins_home/workspace/retry
[Pipeline] {
[Pipeline] timestamps
[Pipeline] {
[Pipeline] stage
[Pipeline] { (Deploy)
[Pipeline] sh
20:38:26 + echo hello
20:38:26 hello
[Pipeline] sleep
20:38:26 Sleeping for 2 sec
[Pipeline] sh
20:38:28 + echo hi
20:38:28 hi
[Pipeline] sleep
20:38:28 Sleeping for 2 sec
[Pipeline] sh
20:38:30 + echo how
20:38:30 how
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // timestamps
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS
```

Without timestamp



## Console Output

```
Started by user kalai
Running in Durability level: MAX_SURVIVABILITY
[Pipeline] Start of Pipeline
[Pipeline] node
Running on Jenkins in /var/jenkins_home/workspace/retry
[Pipeline] {
[Pipeline] timeout
Timeout set to expire in 5 sec
[Pipeline] {
[Pipeline] stage
[Pipeline] { (Deploy)
[Pipeline] sh
+ echo hello
hello
[Pipeline] sleep
Sleeping for 10 sec
Cancelling nested steps due to timeout
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // timeout
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Timeout has been exceeded
Finished: ABORTED
```

## TRIGGER BUILDS

### ❖ Build Triggers

- ❖ Trigger jobs from Pipeline script
- ❖ How to trigger second build, even first build fails
- ❖ How to change build result
- ❖ Call a job by passing parameters

Trigger Other Jobs

- we used build('jobname') option

syntax

```
pipeline {
 agent any

 stages {
 stage('triggerjob') {
 steps {
 build('job1')
 build('job2')
 }
 }
 }
}
```

O/P

### Console Output

```
Started by user kalai
Running in Durability level: MAX_SURVIVABILITY
[Pipeline] Start of Pipeline
[Pipeline] node
Running on Jenkins in /var/jenkins_home/workspace/retry
[Pipeline] {
[Pipeline] stage
[Pipeline] { (triggerjob)
[Pipeline] build (Building job1)
Scheduling project: job1
Starting building: job1 #1
[Pipeline] build (Building job2)
Scheduling project: job2
Starting building: job2 #1
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS
```

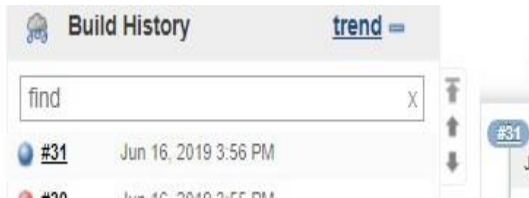
### Trigger second job even first job fails

if we triggering two job, if first job got failed, it wont trigger the second job.so we gng say **propagate:false**, so even though job failed, second job will get trigger.

```
pipeline {
 agent any

 stages {
 stage('triggerjob') {
 steps {
 build(job:'job1', propagate:false)
 build('job2')
 }
 }
 }
}
```

even though job1 failed, its showing succes status.



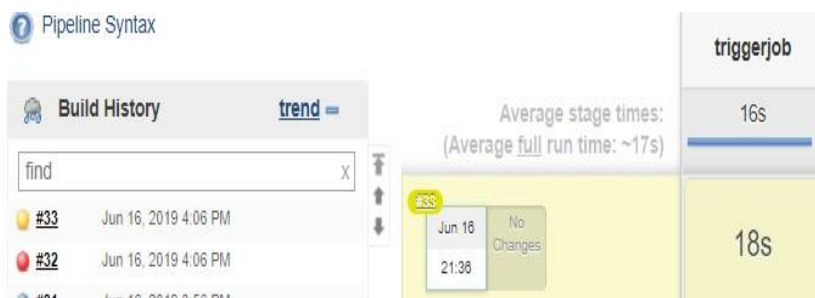
## change build result

while using the below function, it will store the status in jobresult, now eventhough job failed, it will run trigger both job, but it will show unstable result status `jobresult = build(job:'jobname', propagate:false).result` syntax

```
pipeline {
 agent any

 stages {
 stage('triggerjob') {
 steps {
 script{
 jobresult = build(job:'job1', propagate:false).result
 build('job2')
 if(jobresult=='FAILURE'){
 currentBuild.result='UNSTABLE'
 }
 }
 }
 }
 }
}
```

O/P



## Trigger other job with parameters

Already job is created, it contains parameter data to build job

## Running job pipeline script

```
pipeline {
 agent any
 parameters {
 choice(
 name:
 'Nodes',
 choices:"Linux\nMac",
 description: "Choose Node!")
 }
}
```

```

 choice(
name: 'Versions',
choices:"3.4\n4.4",
 description: "Build for which version?")
string(
 name: 'Path',
 defaultValue:"/home/pencillr/builds/",
description: "Where to put the build!")
 } stages {
stage("build") {
steps {
script {

```

```

 echo "$Nodes"
 echo "Versions"
 echo "Path"

```

**triggering job pipeline script:**

```

pipeline {
agent any

 stages {
stage("build") {
steps {
script {

 build(job: "builder-job",
 parameters:
 [string(name: 'Nodes', value: "Linux"),
string(name: 'Versions', value: "3.4"), string(name:
'Path', value: "/home/pencillr/builds/{")])

 }
 }
 }
}
}

```

## Schedule Jobs

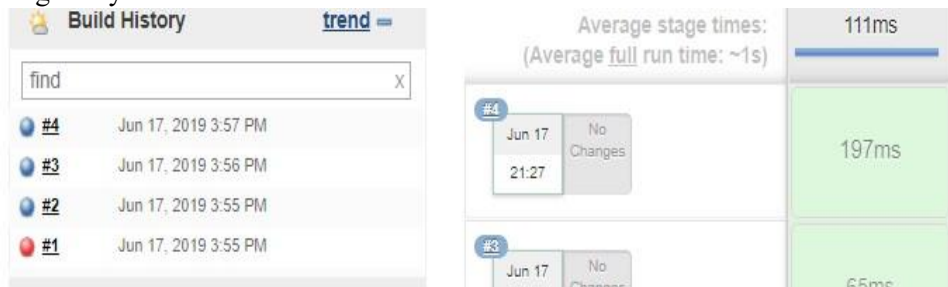
### Schedule Jobs

- ❖ Cron
- ❖ Poll SCM

**Cron** - trigger job will run depends up the cron schedule

```
pipeline {
 agent
 any options {
 timestamps()
 }
 triggers {
 cron('* * * * *')
 }
 stages {
 stage("cron") {
 steps {
 echo "heloo"
 }
 }
 }
}
```

job is running every min



**Poll SCM**- will trigger the job depends up the changes in code, if there is no commit it wont run.

```
pipeline {
 agent
 any options {
 timestamps()
 }
 triggers {
 pollSCM('* * * * *')
 }
 stages {
 stage("cron") {
 steps {
 echo "heloo"
 git
 url:"https://github.com/kalaiarasan33/public.git"
 }
 }
 }
}
```



### Pipeline poll

Recent Changes

Stage View

Average stage times:  
(Average full run time: ~5s)

Jun 17 21:31 No Changes

```

Print Message -- heloo (self time 14ms)
Git -- https://github.com/kalaiaarasan33/public.git (self time 4s)

21:31:29 No credentials specified
21:31:29 Cloning the remote Git repository
21:31:29 Cloning repository https://github.com/kalaiaarasan33/public.git
21:31:29 > git init /var/jenkins_home/workspace/poll # timeout=10
21:31:30 Fetching upstream changes from https://github.com/kalaiaarasan33/public.git
21:31:30 > git --version # timeout=10
21:31:30 > git fetch --tags --progress https://github.com/kalaiaarasan33/public.git
s/remotes/origin/*
21:31:32 > git config remote.origin.url https://github.com/kalaiaarasan33/public.git
21:31:32 > git config --add remote.origin.fetch +refs/heads/*:refs/remotes/origin/*
21:31:32 > git config remote.origin.url https://github.com/kalaiaarasan33/public.git
21:31:32 Fetching upstream changes from https://github.com/kalaiaarasan33/public.git
21:31:32 > git fetch --tags --progress https://github.com/kalaiaarasan33/public.git
s/remotes/origin/*
21:31:33 > git rev-parse refs/remotes/origin/master^{commit} # timeout=10
21:31:33 > git rev-parse refs/origin/origin/master^{commit} # timeout=10
21:31:33 Checking out Revision 56cbe485e5855ea202ec7c859ca5af47a5c3defd (refs/remotes/origin/master)
21:31:33 > git config core.sparsecheckout # timeout=10
21:31:33 > git checkout -f 56cbe485e5855ea202ec7c859ca5af47a5c3defd

```

## Parallel

### ❖ Parallel:

- ❖ Can I use multiple steps under same stage?
- ❖ How to execute jobs at same time ?
- ❖ How to execute stages parallel
- ❖ What is the use of FailFast

**Multiple steps sections under same stage** No we cant

use multiple steps in same stage.like below

```

stages {
 stage("cron") {
 steps {
 echo "step1"
 }
 steps {
 echo "step2"
 }
 }
}

```

**Parallel builds** -- it will trigger the build parallely

pipeline {

agent any

```

 stages {
 stage("build") {
 parallel{
 stage('job1'){
 steps{
 echo "job1"
 }
 }
 stage('job2'){
 steps{
 echo "job2"
 }
 }
 }
 }
 }
}

```

build Job is triggering parallelly

Stage View



## Parallel stages:

```

pipeline {
 agent any
 options {
 timestamps()
 }

 stages {
 stage("stage1") {
 parallel{
 stage('stage1job1'){
 steps{
 echo "stage1job1"
 sleep(10)
 }
 }
 stage('stage1job2'){
 steps{
 echo "stage1job2"
 sleep(10)
 }
 }
 }
 }
 }
}
stage("stage2") {
 parallel{

```



```

 }
 }
 stage('stage1job2'){
 steps{
 eecho "stage1job2"

 sleep(10) }
 }
 stage('stage2job1'){
 steps{
 echo "stage2job1"
 sleep(5)
 }
 }

 stage('stage2job2'){
 steps{
 echo "stage2job2"
 sleep(5)
 }
 }
}
}
}
}

```

**Stage View**

Average stage times:  
(Average full run time: ~11s)

|                                   | stage1 | stage1job1       | stage1job2      | stage2job1       | stage2job2       |
|-----------------------------------|--------|------------------|-----------------|------------------|------------------|
|                                   | 129ms  | 3s               | 2s              | 1s               | 1s               |
| #13<br>Jun 17 22:20<br>No Changes | 132ms  | 622ms<br>aborted | 396ms<br>failed | 568ms<br>aborted | 553ms<br>aborted |
| #12                               |        |                  |                 |                  |                  |

## POST JOBS

post will execute after the completion of pipeline's stages section contains the following blocks

- ❖ POST :
  - ❖ Always
  - ❖ Changed
  - ❖ Fixed
  - ❖ Regression
  - ❖ Aborted
  - ❖ Failure
  - ❖ Success

- ❖ POST blocks cont...
  - ❖ Unstable
  - ❖ Unsuccessful
  - ❖ Cleanup

Post stage and stages level

**Always** : Runs always, wont depend upon the build result **changed**: Runs only

if current build status is changed when compare to previous

**Fixed**: current status is success and previous status is failed

**Regression**: if current status is fail/unstable/aborted and previous run is successful.

**Aborted**: if the current status is aborted

**Failure** : Runs only if the current build status is failed.

**Success** : current build is success

**Unstable** : current build is unstable

**cleanup** : like always, will execute at every time in the last ( if you want to delete any workspace and cleaup any folder , we can use this)

```

pipeline {
 agent any
 options{
 timestamps()
 }

 stages {

```

```

stage("stage1") {
 steps{
 sh "ls -l"
 }
 post{
 always{
 echo " action always "
 }
 changed{
 echo " action always Changed from previous state"
 }
 fixed{
 echo " action Fixed when previous state is failure"
 }
 regression{
 echo " action when current state is fail/unstable/aborted ,
previous state is
success"
 }
 aborted{
 echo " action always aborted"
 }
 failure{
 echo " action always failure"
 }
 success{
 echo " action always success"
 }
 unstable{
 echo " action unstable"
 }
 cleanup{
 echo " action similar like always , it is using to cleanup folder or
workspace"
 }
 }
}

}

}

```

**Previous build is failed, current build success O/P.**

So Always , change (changes in state from previous state ), fixed (previous build failed, current passed), all executed

**Stage Logs (stage1)**

- Shell Script -- ls -l (self time 364ms)
- Print Message -- print always action (self time 32ms)
- Print Message -- print always Changed (self time 370ms)
- Print Message -- print always Fixed (self time 125ms)

cent.Changes

ew

stage1

Average stage times: 821ms

age full run time: ~2s)

No Changes

No Changes

No Changes

571ms

1s

1s

Failed

**Previous build is success O/P**

So always only executed, there no action for change and fixed

**Stage Logs (stage1)**

- Shell Script -- ls -l (self time 326ms)
- Print Message -- print always action (self time 40ms)

ne post

cent.Changes

iew

stage1

Average stage times: 821ms

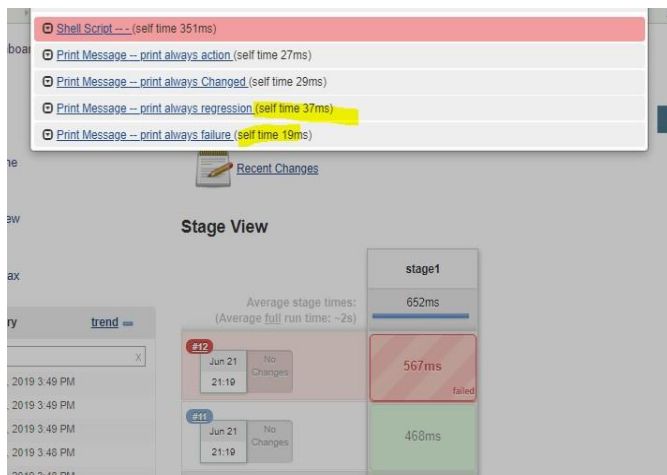
age full run time: ~2s)

No Changes

No Changes

571ms

1s



## Console Output

```
Started by user kalai
Running in Durability level: MAX_SURVIVABILITY
[Pipeline] Start of Pipeline
[Pipeline] node
Running on Jenkins in /var/jenkins_home/workspace/post
[Pipeline] {
[Pipeline] timestamps
[Pipeline] {
[Pipeline] stage
[Pipeline] { (stage1)
[Pipeline] sh
21:21:58 + ls
[Pipeline] sleep
21:21:58 Sleeping for 5 sec
Aborted by kalai
Post stage
[Pipeline] echo
21:22:00 print always action
[Pipeline] echo
21:22:00 print always Changed
[Pipeline] echo
21:22:00 print always regression
[Pipeline] echo
21:22:00 print always aborted
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // timestamps
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: ABORTED
```

## TOOLS

If you want to run specific version of tools to use in pipeline for specific job, so we using tools.

Ex: maven in two version





```

pipeline{
 agent any
 tools{
 maven 'Maven3.6.1'
 }
 stages{
 stage('tools_version'){
 steps{
 sh 'mvn --version'
 }
 }
 }
}

```

O/P

### Console Output

```

Started by user kalai
Running in Durability level: MAX_SURVIVABILITY
[Pipeline] Start of Pipeline
[Pipeline] node
Running on Jenkins in /var/jenkins_home/workspace/Tools
[Pipeline] { (hide)
[Pipeline] stage
[Pipeline] { (Declarative: Tool Install)
[Pipeline] tool
[Pipeline] envVarsForTool
[Pipeline] }
[Pipeline] // stage
[Pipeline] withEnv
[Pipeline] {
[Pipeline] stage
[Pipeline] { (tools_version)
[Pipeline] tool
[Pipeline] envVarsForTool
[Pipeline] withEnv
[Pipeline] {
[Pipeline] sh
+ mvn --version
Apache Maven 3.6.1 (d66c9c8b3152b2e69ee9bac180bb8fcc8e6af555; 2019-04-04T19:00:29Z)
Maven home: /var/jenkins_home/tools/hudson.tasks.Maven_MavenInstallation/Maven3.6.1
Java version: 1.8.0_212, vendor: Oracle Corporation, runtime: /usr/local/openjdk-8/jre
Default locale: en, platform encoding: UTF-8
OS name: "linux", version: "4.15.0-51-generic", arch: "amd64", family: "unix"
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS

```

## Different version maven in job

```

pipeline{
 agent any
 tools{
 maven 'Maven3.5.0'
 }
 stages{
 stage('tools_version'){
 steps{
 sh 'mvn --version'
 }
 }
 }
}

```

## Console Output

```
Started by user kalai
Running in Durability level: MAX_SURVIVABILITY
[Pipeline] Start of Pipeline
[Pipeline] node
Running on Jenkins in /var/jenkins_home/workspace/Tools
[Pipeline] {
[Pipeline] stage
[Pipeline] { (Declarative: Tool Install)
[Pipeline] tool
[Pipeline] mvn --version
[Pipeline] sh
+ mvn --version
Apache Maven 3.5.0 (ff8f5e7444045639af65f6895c6210b5713f428; 2017-04-03T19:39:06Z)
Maven home: /var/jenkins_home/tools/hudson.tasks.Maven_MavenInstallation/Maven3.5.0
Java version: 1.8.0_212, vendor: Oracle Corporation
Java home: /usr/local/openjdk-8/jre
Default locale: en, platform encoding: UTF-8
OS name: "linux", version: "4.15.0-51-generic", arch: "amd64", family: "unix"
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS
```

Tools in Stage level :

```
pipeline{ agent any
```

```
tools{ maven
```

```
'Maven3.6.1'
```

```
} stages{
```

```
stage('tools_version'){
```

```
steps{
```

```
sh 'mvn --version'
```

```
}
```

```
}
```

```
stage('diff_version_stage_level')
```

```
{ tools{ maven
```

```
'Maven3.5.0'
```

```
}
```

```
steps{ echo "stage level"
```

```
sh 'mvn --version'
```

```
}
```

```
}
```

```
}
```

```
}
```

```

Running on Jenkins in /var/jenkins_home/workspace/Tools
[Pipeline] {
[Pipeline] stage
[Pipeline] { (Declarative: Tool Install)
[Pipeline] tool
[Pipeline] envVarsForTool
[Pipeline] }
[Pipeline] // stage
[Pipeline] withEnv
[Pipeline] {
[Pipeline] stage
[Pipeline] { (tools_version)
[Pipeline] tool
[Pipeline] envVarsForTool (hide)
[Pipeline] withEnv
[Pipeline] {
[Pipeline] sh
[Pipeline] mvn --version
Apache Maven 3.6.1 (d66c9c0b3152b2e69ee9bac180bb8fcc8e6af555; 2019-04-04T19:00:29Z)
Maven home: /var/jenkins_home/tools/hudson.tasks.Maven_MavenInstallation/Maven3.6.1
Java version: 1.8.0_212, vendor: Oracle Corporation, runtime: /usr/local/openjdk-8/jre
Default locale: en, platform encoding: UTF-8
OS name: "linux", version: "4.15.0-51-generic", arch: "amd64", family: "unix"
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (diff_version_stage_level)
[Pipeline] tool
[Pipeline] envVarsForTool
[Pipeline] withEnv
[Pipeline] {
[Pipeline] echo
stage level
[Pipeline] sh
+ mvn --version
Apache Maven 3.5.0 (ff8f5e7444045639af65f6095c62210b5713f426; 2017-04-03T19:39:06Z)
Maven home: /var/jenkins_home/tools/hudson.tasks.Maven_MavenInstallation/Maven3.5.0
Java version: 1.8.0_212, vendor: Oracle Corporation
Java home: /usr/local/openjdk-8/jre
Default locale: en, platform encoding: UTF-8
OS name: "linux", version: "4.15.0-51-generic", arch: "amd64", family: "unix"
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // stage
[Pipeline] // withEnv
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS

```

## Conditional and Loop Statements

IF Condition:

We can use Groovy coding functionalities using script {...} section.

```

pipeline{
 agent any
 environment{
 Tools='Jenkins'
 }
 stages{
 stage('conditions'){
 steps{
 script{
 if(Tools == 'Jenkins'){
 echo 'Tools is
 jenkins'
 }else{
 echo 'Tools is not jenkins '
 }
 }
 }
 }
}

```

## Console Output

```

Started by user kalai
Running in Durability level: MAX_SURVIVABILITY
[Pipeline] Start of Pipeline
[Pipeline] node
Running on jenkins in /var/jenkins_home/workspace/Tools
[Pipeline] {
[Pipeline] withEnv
[Pipeline] {
[Pipeline] stage
[Pipeline] { (conditions)
[Pipeline] script
[Pipeline] {
[Pipeline] echo
Tools is jenkins
[Pipeline] }
[Pipeline] // script
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS

```

Demo : check build number even or Odd

```

pipeline{
 agent any
 environment{
 Tools='Jenkins'
 }
 stages{
 stage('conditions'){
 steps{
 script{
 int
 buildno="$BUILD_NUMBER"
 if(buildno %2 == 0){ echo 'buildno
 is even'
 }else{
 echo 'buildno is odd'
 }
 }
 }
 }
 }
}

```



Demo: For loop

```

pipeline{
 agent any
 environment{
 Tools='Jenkins'
 }
 stages{
 stage('conditions'){
 steps{
 script{

```

```

for(i=0;i<=5;i++){
println i

}

 int
 buildno="$BUILD_NUMBER"
 if(buildno %2 == 0){ echo 'buildno
is even'
 }else{
 echo 'buildno is odd'
 }
 }
}
}
}

```



### Other Example

- ❖ Other examples
  - ❖ Ansi color
  - ❖ Change BuildName and Description
  - ❖ Dir, cleanws
  - ❖ Write file jenkins syntax
  - ❖ Maven example
  - ❖ Archive artifacts
  - ❖ Finger prints
- ❖ Other examples cont...
  - ❖ Credentials
  - ❖ Check OS type
  - ❖ Trim string

### AnsiColor:

we need to install the plugin first , then set the ansi in configuration ,jenkins foreground

```

pipeline{
 agent any
 stages{
 stage('ansi'){
 steps{
 ansiColor('xterm') {
 echo 'something that outputs ansi colored stuff'
 }
 }
 }
 stage('non_ansi'){
 steps{
 echo 'non_ansi'
 }
 }
 }
}

```

| Custom color maps | Name               | xterm           |
|-------------------|--------------------|-----------------|
|                   | Default Background | Jenkins Default |
|                   | Default Foreground | Green           |
|                   | Black              | #000000         |
|                   |                    | #4C4C4C         |
|                   | Red                | #CD0000         |
|                   |                    | #FF0000         |
|                   | Green              | #00CD00         |
|                   |                    | #00FF00         |
|                   | Yellow             | #CDCD00         |
|                   |                    | #FFFF00         |
|                   | Blue               | #1E90FF         |
|                   |                    | #4682B4         |
|                   | Magenta            | #CD00CD         |
|                   |                    | #FF00FF         |
|                   | Cyan               | #00CDCD         |
|                   |                    | #00FFFF         |
|                   | White              | #E0E0E0         |
|                   |                    | #FFFFFF         |

## Console Output

```

Started by user kalai
Running in Durability level: MAX_SURVIVABILITY
[Pipeline] Start of Pipeline
[Pipeline] node
Running on Jenkins in /var/jenkins_home/workspace/Example/ansi
[Pipeline] {
[Pipeline] stage
[Pipeline] { (ansi)
[Pipeline] ansiColor
[Pipeline] {
[Pipeline] echo
[Pipeline] something that outputs ansi colored stuff
[Pipeline] }
[Pipeline] // ansiColor
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (non_ansi)
[Pipeline] echo
[Pipeline] non_ansi
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS

```

## Change Build Number to Name

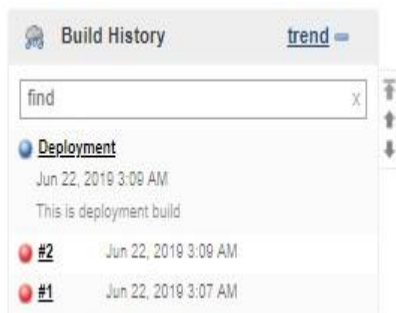
This is used to define the name for the job and description.

```

pipeline{
 agent any
 stages{
 stage('buid_name'){
 steps{
 script{
 currentBuild.displayName = "Deployment"
 currentBuild.description = "This is deployment build"
 }
 echo 'build name changing'
 }
 }
 }
}

```

O/P



**dir, cleanws** create folder inside workspace -> job name -> (creating folder) -> job

output is here

```

kalai@jenlinux:~/jenkins_home/workspace/Example$ cd delete_ws/ --> job name
kalai@jenlinux:~/jenkins_home/workspace/Example/delete_ws$ ls build_one
build_one@tmp --> folder we created with dir function
kalai@jenlinux:~/jenkins_home/workspace/Example/delete_ws$ cd build_one
kalai@jenlinux:~/jenkins_home/workspace/Example/delete_ws/build_one$ ls
hello.txt --> output created
kalai@jenlinux:~/jenkins_home/workspace/Example/delete_ws/build_one$ pwd
/home/kalai/jenkins_home/workspace/Example/delete_ws/build_one Creating
output in workspace -> jobname -> build_one --> outputfiles

```

```

pipeline{
 agent any
 stages{
 stage('cleanWS'){
 steps{ dir('build_one'){ script{
 currentBuild.displayName = "Deployment"
 currentBuild.description = "This is deployment build"
 }
 sh "echo dir creation and delete WS > hello.txt"
 }
 }
}
}

```

Creating output in workspace -> jobname -> build\_one --> outputfiles --> deleted job workspace

```
kalai@jenlinux:~/jenkins_home/workspace/Example/delete_ws/build_one/..$ cd ..
kalai@jenlinux:~/jenkins_home/workspace/Example$ ls change_build_name
change_build_name@tmp kalai@jenlinux:~/jenkins_home/workspace/Example$
```

```
pipeline{
 agent any
 stages{
 stage('cleanWS'){
 steps{ dir('build_one'){ script{
 currentBuild.displayName = "Deployment"
 currentBuild.description = "This is deployment build"
 }
 sh "echo build name changing > hello.txt"
 }
 cleanWs()
 }
}
```

### Console Output

```
Started by user kalai
Running in Durability level: MAX_SURVIVABILITY
[Pipeline] Start of Pipeline
[Pipeline] node
Running on Jenkins in /var/jenkins_home/workspace/Example/delete_ws
[Pipeline] {
[Pipeline] stage
[Pipeline] { (cleanWS)
[Pipeline] dir
Running in /var/jenkins_home/workspace/Example/delete_ws/build_one
[Pipeline] {
[Pipeline] script
[Pipeline] {
[Pipeline] }
[Pipeline] // script
[Pipeline] sh
+ echo build name changing
[Pipeline] }
[Pipeline] // dir
[Pipeline] cleanWs
[WS-CLEANUP] deleting project workspace...
[WS-CLEANUP] Deferred wipeout is used...
[WS-CLEANUP] done
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS
```

## Write file\_Jenkins syntax

Creating file in jenkins syntax

```
pipeline{
 agent any
 stages{
 stage('write_file'){
 steps{
 writeFile file: 'newfile.txt' , text:"my file content is very small"
 archiveArtifacts '*.txt'
 }
 }
 }
}
```





## Sample Maven Build

Build the maven project

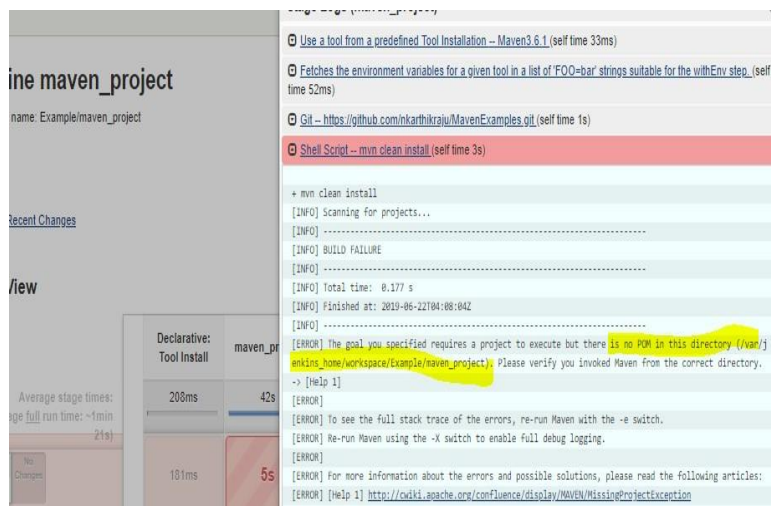
```

pipeline{
 agent any
 tools{
 maven 'Maven3.6.1'
 }
 stages{
 stage('maven_project'){
 steps{
 git url:"https://github.com/nkarthikraju/MavenExamples.git"
 sh "mvn clean install"
 }
 }
 }
}

```

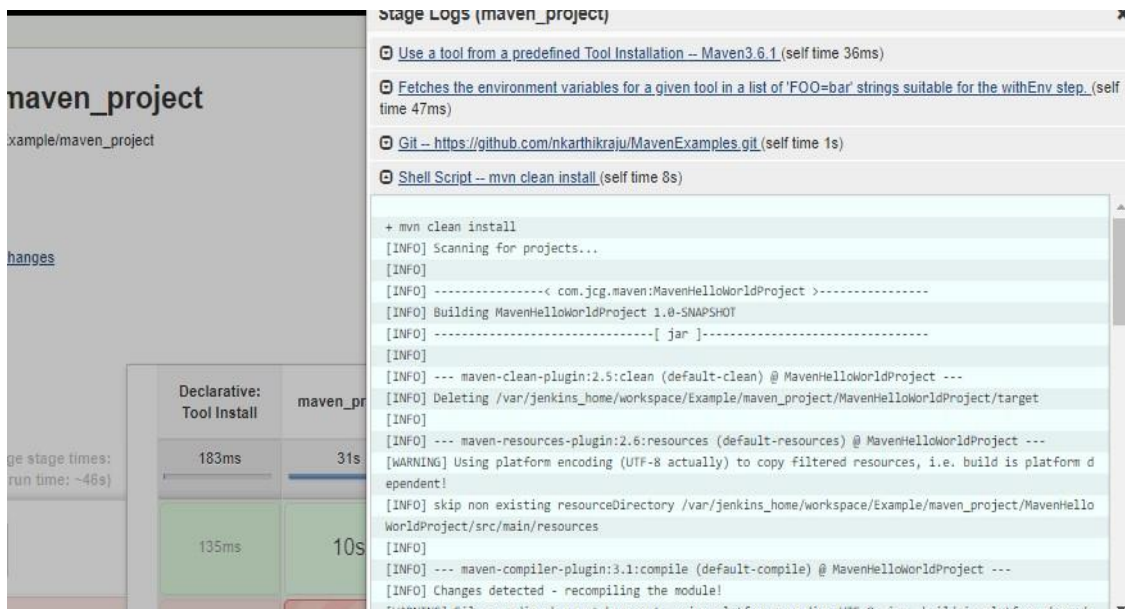
**without moving into the directory , it will show error , no pom**

file



now moved to directory with help of dir function then executing maven clean install

```
pipeline{
 agent any
 tools{
 maven 'Maven3.6.1'
 }
 stages{
 stage('maven_project'){
 steps{
 git url:"https://github.com/nkarthikraju/MavenExamples.git"
 dir('MavenHelloWorldProject'){ sh "mvn clean install"
 }
 }
 }
 }
}
```



The screenshot displays the Jenkins Pipeline console output for the 'maven\_project' stage. On the left, a summary table shows the stage's performance:

| Stage          | Declarative: Tool Install | maven_pr |
|----------------|---------------------------|----------|
| Stage times:   | 183ms                     | 31s      |
| Run time: ~46s | 135ms                     | 10s      |

The main console output shows the following steps and their logs:

- Use a tool from a predefined Tool Installation -- Maven3.6.1** (self time 36ms)
- Fetches the environment variables for a given tool in a list of 'FOO=bar' strings suitable for the withEnv step.** (self time 47ms)
- Git -- https://github.com/nkarthikraju/MavenExamples.git** (self time 1s)
- Shell Script -- mvn clean install** (self time 8s)

The shell script logs show the execution of 'mvn clean install' with various Maven build messages, including scanning for projects, deleting the target directory, and compiling the module.

## Archive artifacts and finger prints

getting archiveArtifacts when build is success

```
pipeline{
 agent any
 tools{
 maven 'Maven3.6.1'
 }
 stages{
 stage('maven_project'){
 steps{
 git url:"https://github.com/nkarthikraju/MavenExamples.git"
 dir('MavenHelloWorldProject'){ sh "mvn clean install"
 }
 }
 }
 }
}
```

```

 }
 post{
 success{
 archiveArtifacts
 "MavenHelloWorldProject/target/*.jar"
 }
 }
}

```

## Build #14 (Jun 22, 2019 4:17:33 AM)



### Fingerprint:

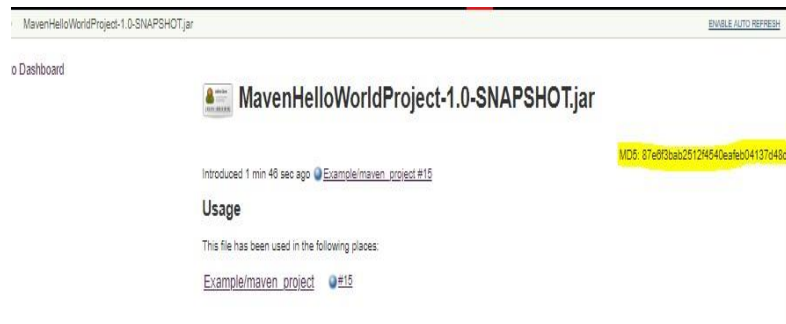
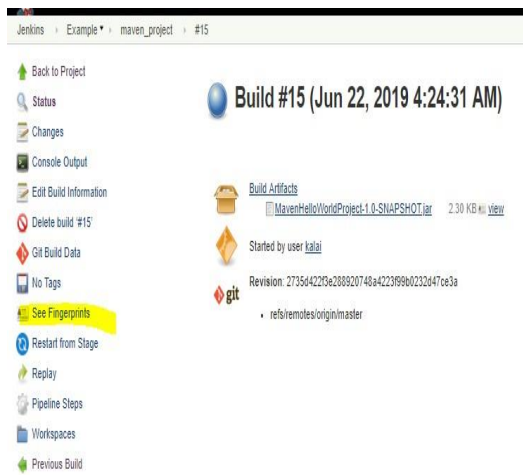
if we execute the same job 10 times or etc, it will give same name output , for the identificaton or record the artifacts with fingerprint it will create checksum with build.

```

pipeline{
 agent any
 tools{
 maven 'Maven3.6.1'
 }
 stages{
 stage('maven_project'){
 steps{
 git url:"https://github.com/nkarthikraju/MavenExamples.git"
 dir('MavenHelloWorldProject'){ sh "mvn clean install"
 }
 }
 }
 post{
 success{
 archiveArtifacts artifacts:"MavenHelloWorldProject/target/*.jar" ,
 fingerprint:true
 }
 }
 }
}

```

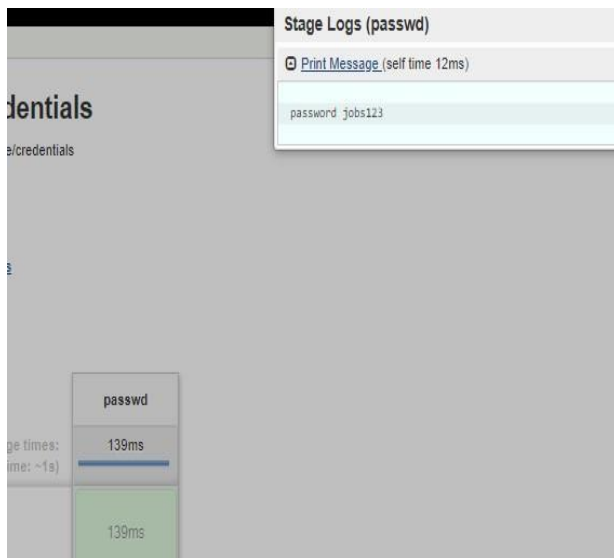
O/P



## Uses of Credentials Option

if we pass the password it will transparent

```
pipeline{
 agent
 any environment{
 pass="jobs123"
 }
 stages{
 stage('passwd'){
 steps{
 echo "password $pass"
 }
 }
 }
}
```



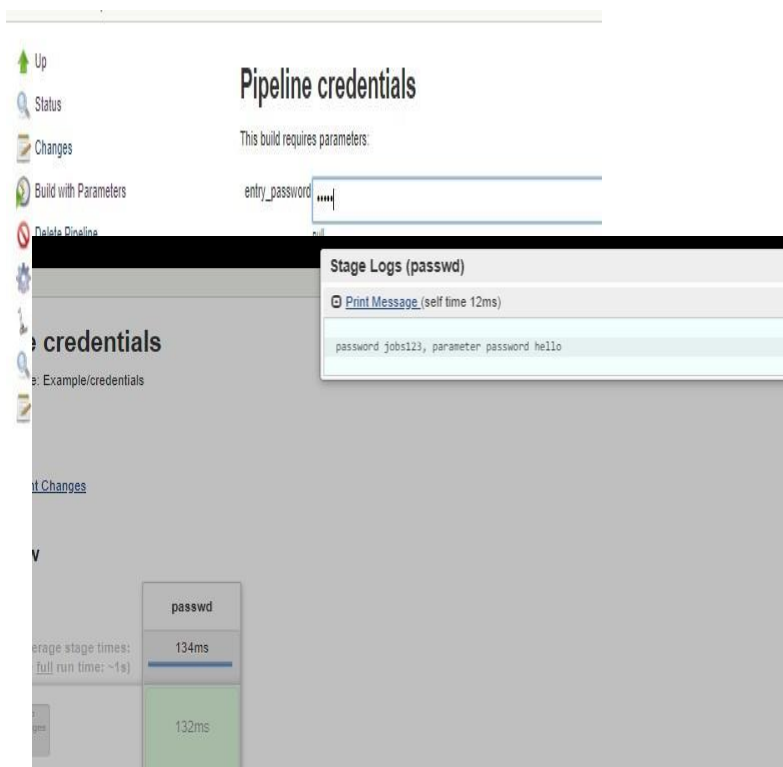
another option passing the password as parameter with (password parameter)

```

pipeline{
 agent
 any environment{
 pass="jobs123"
 }
 parameters{
 password(name:'entry_password')
 }
 stages{
 stage('passwd'){
 steps{
 echo "password $pass, password parameter $entry_password"
 }
 }
 }
}

```

O/P



Now password with credential function

create secret text in credentials in jenkins



## Global credentials (unrestricted)

Credentials that should be available irrespective of domain specification to requirements matching.

|  | Name                              |
|--|-----------------------------------|
|  | <a href="#">remote_user</a>       |
|  | <a href="#">DB_PASSWORD</a>       |
|  | <a href="#">AWS_ACCESS_KEY_ID</a> |
|  | <a href="#">TEST_PASS</a>         |

Icon: [S](#) [M](#) [L](#)

```
pipeline{
 agent any
 environment{
 pass="jobs123"
 password=credentials('DB_PASSWORD')
 }
 parameters{
 password(name:'entry_password')
 }
 stages{
 stage('passwd'){
 steps{
 echo "password $pass, parameter password $entry_password , credential funtion password $password"
 }
 }
 }
}
```



## CheckOS\_AndExecuteSteps

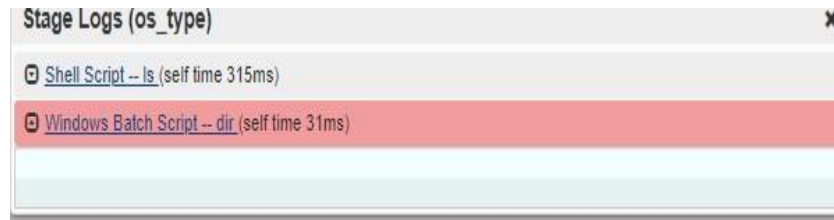
if we execute like below it will show error, so we need to check the os type with of function

```
pipeline{
 agent any
```

```

 stages{
 stage('os_type'){
 steps{
 sh "ls"
 bat "dir"
 }
 }
 }
}

```



now it will check the ostype and then it will execute

```

pipeline{
agent any

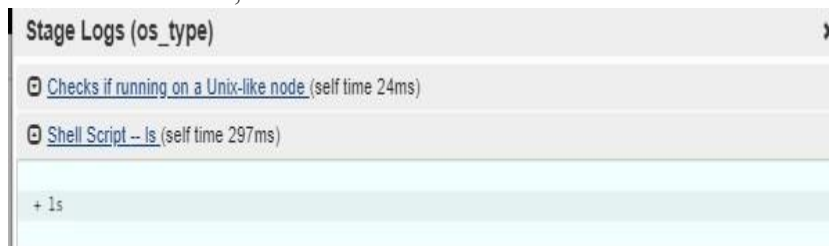
```

```

 stages{
 stage('os_type'){
 steps{
 script{
 if(isUnix()){
 sh "ls"
 }else{
 bat
 "dir"
 }
 }
 }
 }
}

```

This is linux machine, so linux command executed



**Trim**

```

pipeline{
agent any
environment{

```

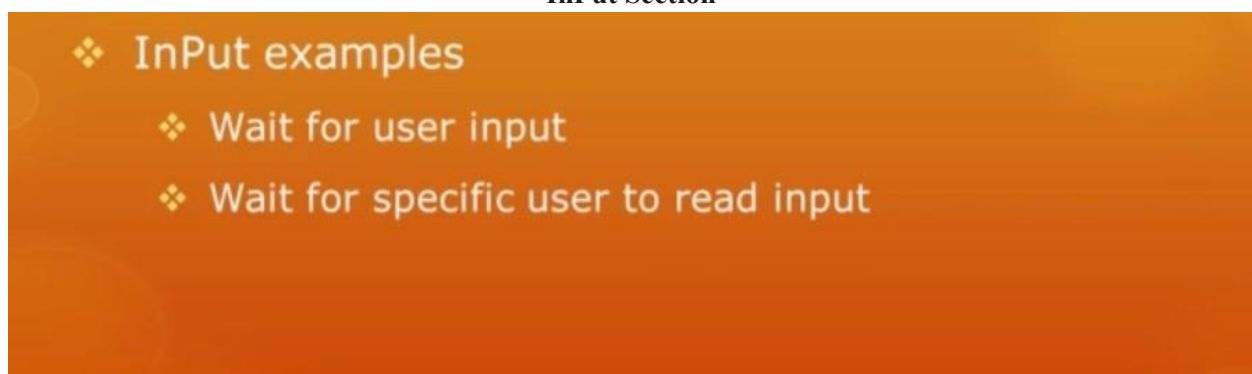
```

 tools='jenkins'
 }
 stages{
stage('trimming_string'){
 steps{ script{
t1=tools[0..6] //jenkins
t2=tools[3..5] //kin
echo "$t1 , $t2"
 }
 }
}
}
}

```



## InPut Section



## Install the plugin



```

pipeline {
agent any

 stages {
stage('build user') {
 steps { wrap([${class:
'BuildUser'}) {
sh 'echo "${BUILD_USER}'"
 }
 }
}
}

```



```
}
}
}
```

it will pull the user name



## Build in specific user

```
pipeline{ agent any stages{ stage('user_input'){
 steps{ wrap([$class: 'BuildUser']){
 script{
 def name1="${BUILD_USER}"
 echo "${BUILD_USER}, $name1"
 if(name1=='kalai'){
 echo "only kalai can able to build"
 }else{
 echo "others cant able to build"
 }
 }
 }
 }
}
}
```



## User input procced or abort

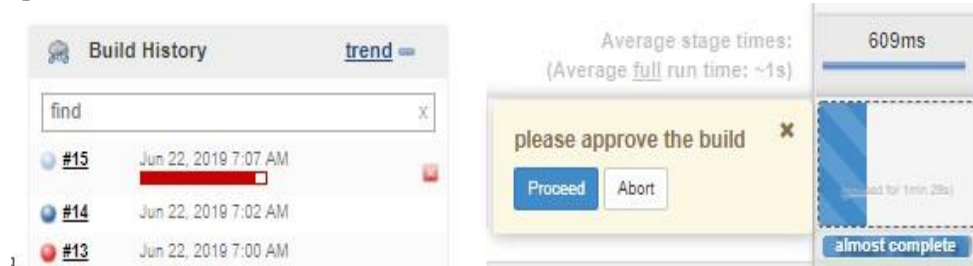
```
pipeline{ agent any stages{
 stage('user_input'){
 steps{
 input("please approve the build")
 script{
 sh "echo this is kalai"
 }
 }
}
```

```

 }
 }
}

```

user I/P proceed or abort



if proceed it will start

### Console Output

```

started by user kalai
Running in Durability level: MAX_SURVIVABILITY
[Pipeline] Start of Pipeline
[Pipeline] node
Running on Jenkins in /var/jenkins_home/workspace/Example/user_in
[Pipeline] {
[Pipeline] stage
[Pipeline] { (user_input)
[Pipeline] input
please approve the build
Proceed or Abort
Approved by kalai
[Pipeline] script
[Pipeline] {
[Pipeline] sh
+ echo this is kalai
this is kalai
[Pipeline] }
[Pipeline] // script
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS

```

### Read Input From Specific user:

we can give list of user permission to proceed, other cant give proceed and get specific string from submitter.

```

pipeline{ agent any
 stages{
 stage('user_input'){
 input{ message "press ok to continue"
 submitter "kalai,lead"
 parameters{
 string(name:'username1',description: "only kalai and lead has permission")
 }
 }
 }
 }
 steps{

 echo "User : ${username1} said ok"
 }
}

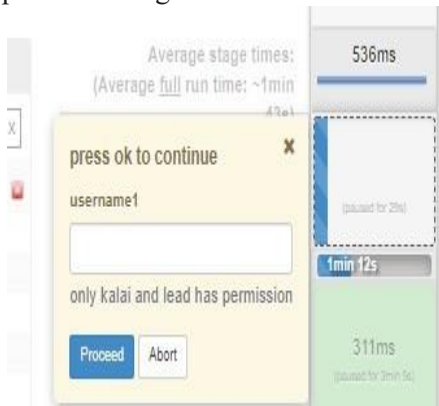
```

```

 }
}
}
}
}

```

can pass the string:



here is said manager

### Console Output

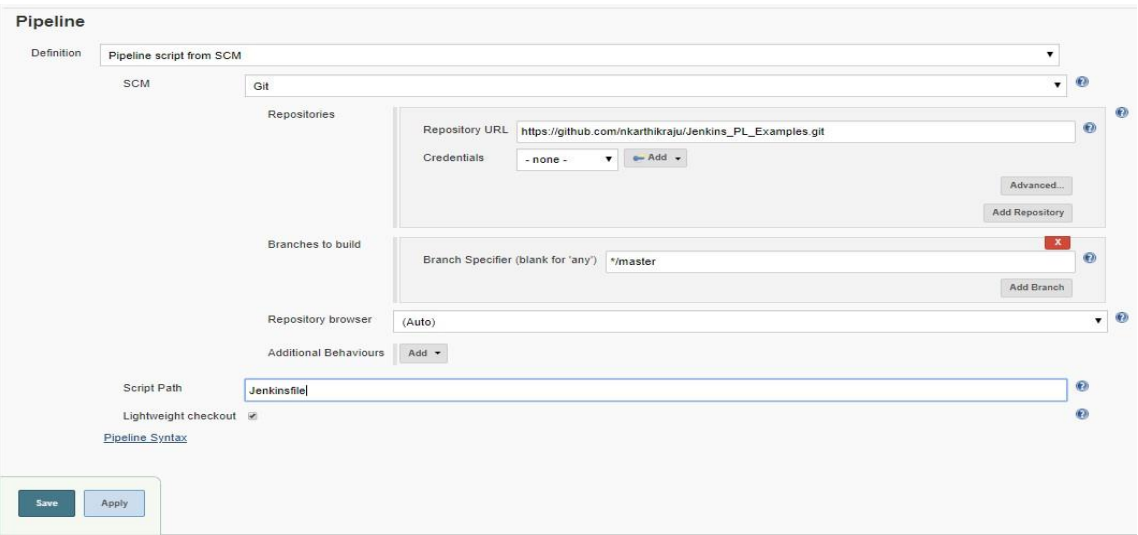
```

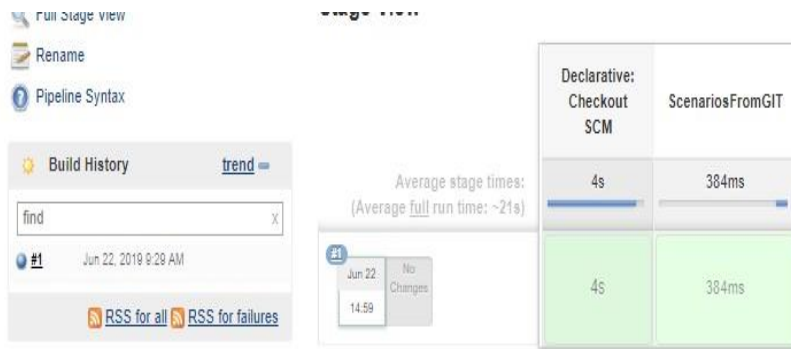
Started by user kalai
Running in Durability level: MAX_SURVIVABILITY
[Pipeline] Start of Pipeline
[Pipeline] node
Running on Jenkins in /var/jenkins_home/workspace/Example/user_in
[Pipeline] {
[Pipeline] stage
[Pipeline] { (user_input)
[Pipeline] input
Input requested
Approved by kalai
[Pipeline] withEnv
[Pipeline] {
[Pipeline] echo
User : manager said ok
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS

```

### SCM GIT

We can check out the jenkinsfile in scm whether it is git or SVN.and can maintain with version control.





## GIT checkout with the help of Pipeline syntax option

We can generate the pipeline code from pipeline syntax option



We can generate the pipeline syntax from passing the parameter to plugin , then with generate option

**Overview**

This Snippet Generator will help you learn the Pipeline Script code which can be used to define various steps. Pick a whole statement into your script, or pick up just the options you care about. (Most parameters are optional and can be changed)

**Steps**

Sample Step:

Repository URL:

Branch:

Credentials:

☒ Include in polling?

☒ Include in changelog?

`git 'https://github.com/nkarthikraju/Jenkins_PL_Examples.git'`

## Commit Code

This script will read the file and write and push to the git

```
pipeline{
 agent any
 stages{
 stage('commitcode'){
 steps{
```



```
sh """
```

```
git add file.txt
```

```
git commit -m "updating with build num committed"
```

```
git push
```

```
"""
```

```
}
```

```
}
```

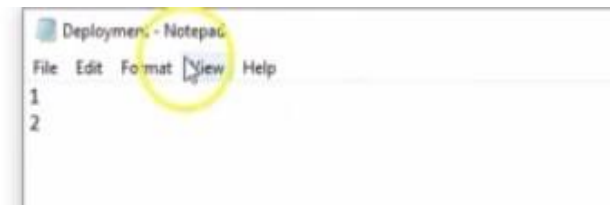
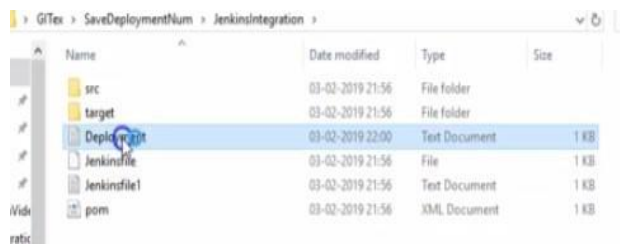
```
}
```

```
}
```

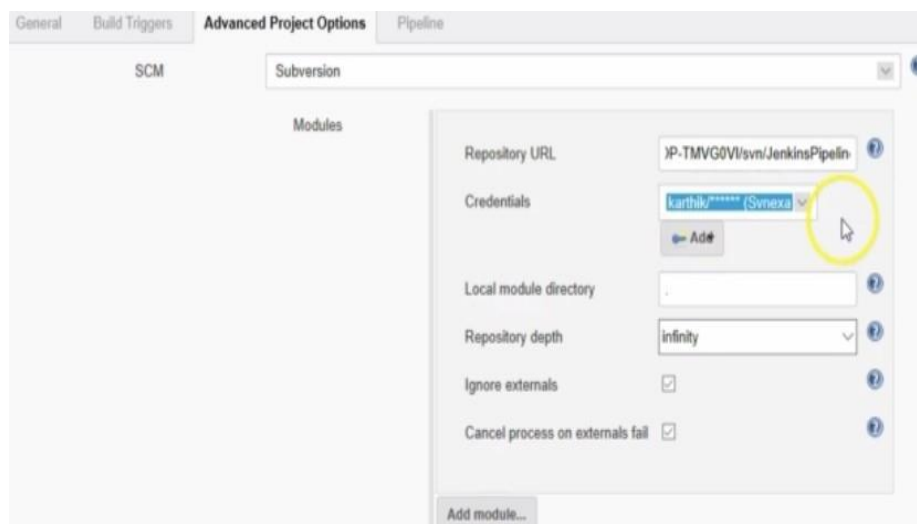
```
}
```

O/P

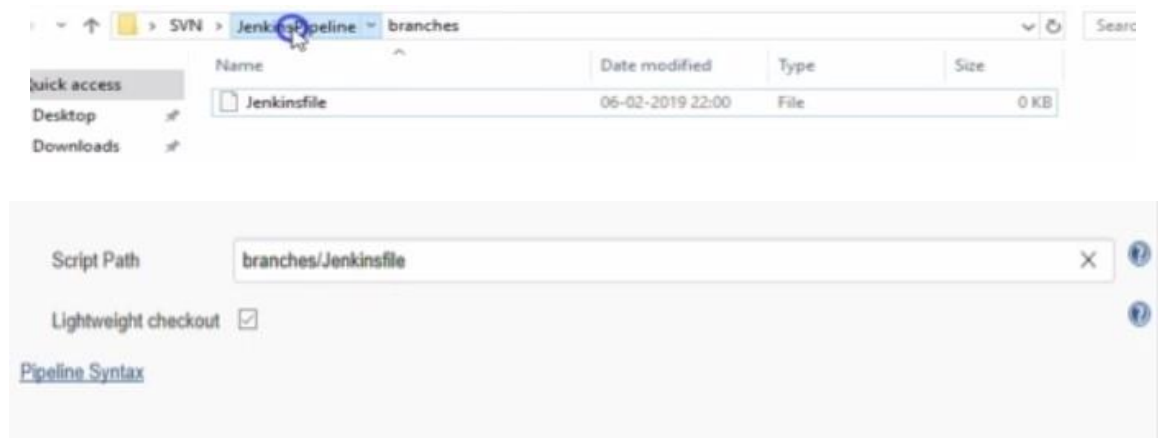
Build no will add in the deployment file



SVN examples



jenkins file in brancher folder



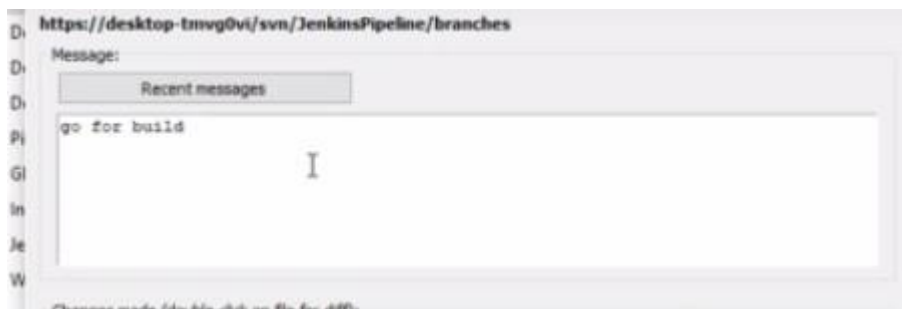
#### Stages based on When condition

```

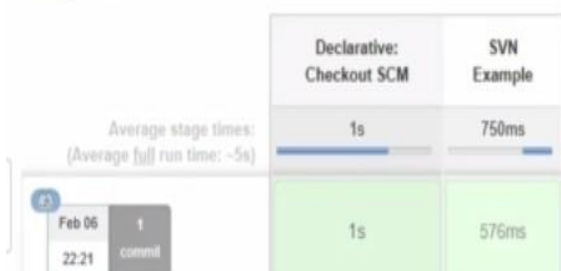
pipeline{
 agent any
 stages{
 stage('svn'){
 when{
 changelog 'build'
 }
 steps{
 sh '''
 echo "found build keyword in the commit, so proceeding further stage"
 '''
 }
 }
 }
}

```

If you commit with build message, then it will build



#### Stage View



#### WHEN

When is similar to If condition, but its more advanced with in-built condition

## ❖ WHEN

- ❖ Run the step when Previous build is Success/Fail
- ❖ Run the step based on the Environment value
- ❖ Changelog
- ❖ Changeset
- ❖ Equals
- ❖ NOT
- ❖ Allof
- ❖ Anyof
- ❖ File exists
- ❖ When file contains data

**Equals and not Equals:**

```
pipeline{
 agent any
 environment{
 Tool="Jenkins"
 } stages{ stage('When_equals'){
 when{
 equals expected:'Docker' , actual: "$Tool"
 }
 steps{
 sh """
 echo " if when equal "
 """
 }
 }
 stage('When_no_equals'){
 when{
 not {
 environment name:Tool , value:"Jenkins"
 }
 }
 steps{
 sh """
 echo " if when not equal "
 """
 }
 }
}
```

**O/P equal expected is not matched , not equals is matched**



## Stage View



O/P equal expected is matched , not equals is matched.

```

pipeline{
 agent any
 environment{
 Tool="Jenkins"
 } stages{ stage('When_equals'){
 when{
 equals expected:'Jenkins' , actual: "$Tool"
 }
 steps{
 sh """
 echo " if when equal "
 """
 }
 }
 stage('When_no_equals'){
 when{
 not {
 environment name:Tool , value:"Jenkins"
 }
 }
 steps{
 sh """
 echo " if when not equal "
 """
 }
 }
}

```

## Stage View

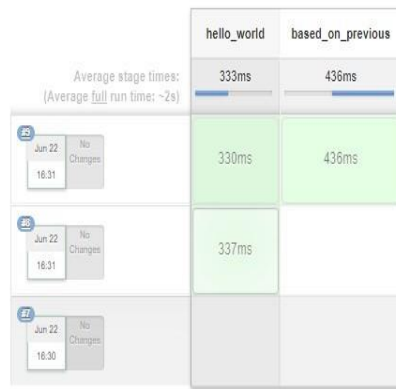


Check previous build result and execute steps

execute when previous build is success

```
pipeline{
 agent any
 environment{
 Tool="Jenkins"
 }
 stages{
 stage('hello_world'){
 steps{
 sh """
 echo "hello_world "
 """
 }
 }
 stage('based_on_previous'){
 when{
 expression {
 currentBuild.getPreviousBuild().result == 'SUCCESS'
 }
 }
 steps{
 sh """
 echo "previous build is failed, now success"
 """
 }
 }
 }
}
```

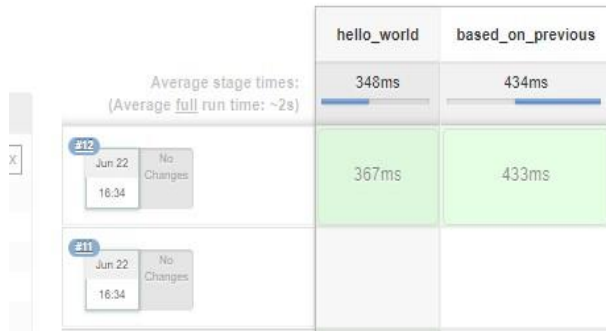
## stage view



When previous build is failure

```
pipeline{
 agent any
 environment{
 Tool="Jenkins"
 }
 stages{
 stage('hello_world'){
 steps{
 sh """
 echo " hello_world "
 """
 }
 }
 stage('based_on_previous'){
 when{
 expression {
 currentBuild.getPreviousBuild().result == 'FAILURE'
 }
 }
 steps{
 sh """
 echo " previous build is failled, now sucess"
 """
 }
 }
 }
}
```

## Stage View



### Steps based on commit messages (jenkinsfile : SCM)

when they committed with message build, it will get execute

```
pipeline{
 agent any
 stages{
 stage('svn'){
 when{
 changelog 'build'
 }
 steps{
 sh """
 echo "found build keyword in the commit, so proceeding further stage "
 """
 }
 }
 }
}
```

### Steps based on Committed files (jenkinsfile : SCM)

```
pipeline{
 agent any
 stages{
 stage('pythonfile'){
 when{
 changeset '*.py'
 }
 steps{
 sh """
 echo "commit files contains only for python file "
 """
 }
 }
 stage('java_file'){
 when{
 changeset '*.xml'
 }
 }
 }
}
```

```
steps{
 sh """
 echo "commit files contains only for xml file" """
}
}
```

- When{
  - allOf{
    - A condition and
    - B condition
  - }
  - anyOf{
    - A condition or
    - B condition
  - }

```
agent any
environment{
 Tool="jenkins"
 envv="PRD"
}

stages{
 stage('allof'){
 when{
 allOf{
 equals expected: "jenkins" , actual: "$Tool"
 equals expected: "PRD" , actual: "$envv"
 }
 }
 }

 steps{

 sh """
 echo " when both condition is pass "
 """
 }
}

stage('anyof'){
 when{ anyOf{
 equals expected: "jenkins" , actual: "$Tool"
 equals expected: "STG" , actual: "$envv"
 }
}

steps{

 sh """
```

```
echo " when any one condition is pass"
```

```
"""
```

```
}
```

```
}
```

```
}
```

```
}
```

## Stage View



### Execute stage if required string is matched in the file

```
pipeline{
 agent any

 stages{
 stage('string_in_file'){
 when{
 expression{ return readFile('C:\\Users\\user\\Desktop\\cloudguru_sysops\\files.txt').contains('truncated')} }

 steps{
 sh """
 echo " string in file "
 """
 }
 }
 }
}
```

### Skip Stage always

if you want to skip the stage or skip for few days.

```
pipeline{
 agent any

 stages{
 stage('string_in_file'){
 when{
 return false
 expression{ return readFile("C:\\Users\\user\\Desktop\\cloudguru_sysops\\files.txt").contains('truncated')}
 }

 steps{
 sh """
 echo " string in file "
 """
 }
 }
 }
}
```

```

 }
 }
}

```

## SHELL

### Shell Syntax and Commands

```

pipeline{
 agent any
 stages{
 stage('shell'){

 steps{

 sh """
 ls -l
 pwd
 """

 }
 }
 }
}

```



**Stage Logs (shell)**

Shell Script -- `ls -l pwd` (self time 274ms)

```

+ ls -l
total 0
+ pwd
/var/jenkins_home/workspace/when

```

### Create file with Build Number and Build Name

```


pipeline{
 agent any
 stages{
 stage('shell'){

 steps{

 sh """
 touch ${JOB_NAME}.${BUILD_NUMBER}.txt
 ls -l pwd
 """

 }
 }
 }
}

```



**Stage Logs (shell)**

Shell Script -- `touch When.30.txt ls -l pwd` (self time 282ms)

```

+ touch When.30.txt
+ ls -l
total 0
-rw-r--r-- 1 jenkins jenkins 0 Jun 22 12:45 When.30.txt
+ pwd
/var/jenkins_home/workspace/when

```

Create Html and Copy to the location.

```
pipeline{
 agent any

 stages{
 stage('html'){
 steps{
 bat """
 echo hello this is my HTML >> "D:\\"
 """
 }
 }
 stage('copy_location'){
 steps{
 bat """ copy *.html
 D:\tomcat\
 """
 }
 }
 }
}
```



