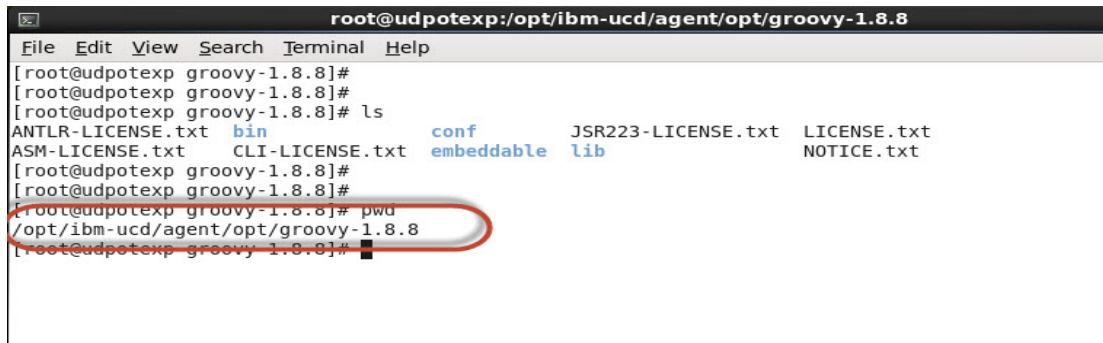


Notes from Wyndham on Groovy scripts w/ UCD

There are different approaches to using groovy, however an easy way to handle various tasks is to use it as a scripting language. All you need is to import the the Groovy plugin (<http://plugins.urbancode.com/uBuild/plugin/Groovy/3.0>). It's easy to get started with writing and testing groovy scripts since there's a groovy environment available wherever there's a UCD agent.



```
root@udpotexp:/opt/ibm-ucd/agent/opt/groovy-1.8.8
File Edit View Search Terminal Help
[root@udpotexp groovy-1.8.8]#
[root@udpotexp groovy-1.8.8]#
[root@udpotexp groovy-1.8.8]# ls
ANTLR-LICENSE.txt  bin          conf          JSR223-LICENSE.txt  LICENSE.txt
ASM-LICENSE.txt    CLI-LICENSE.txt  embeddable    lib                 NOTICE.txt
[root@udpotexp groovy-1.8.8]#
[root@udpotexp groovy-1.8.8]# pwd
/opt/ibm-ucd/agent/opt/groovy-1.8.8
[root@udpotexp groovy-1.8.8]#
```

If you're testing scripts from the command line, you may want to set \$GROOVY_HOME and add \$GROOVY_HOME/bin to your path. Once the script is tested you can paste the script right into the Run Groovy Script step in the process. You can also test scripts in your browser (<http://groovyconsole.appspot.com/>) and you can find examples of groovy scripts other people have written.

I found a few sites fishing around for some general context on groovy

- primer on writing groovy scripts <http://groovy.codehaus.org/Groovy+as+script>
- getting started tutorial <http://groovy.codehaus.org/Tutorial+1+--+Getting+started>

Getting Ready

Before we start with our example, here are a few things you may want to do:

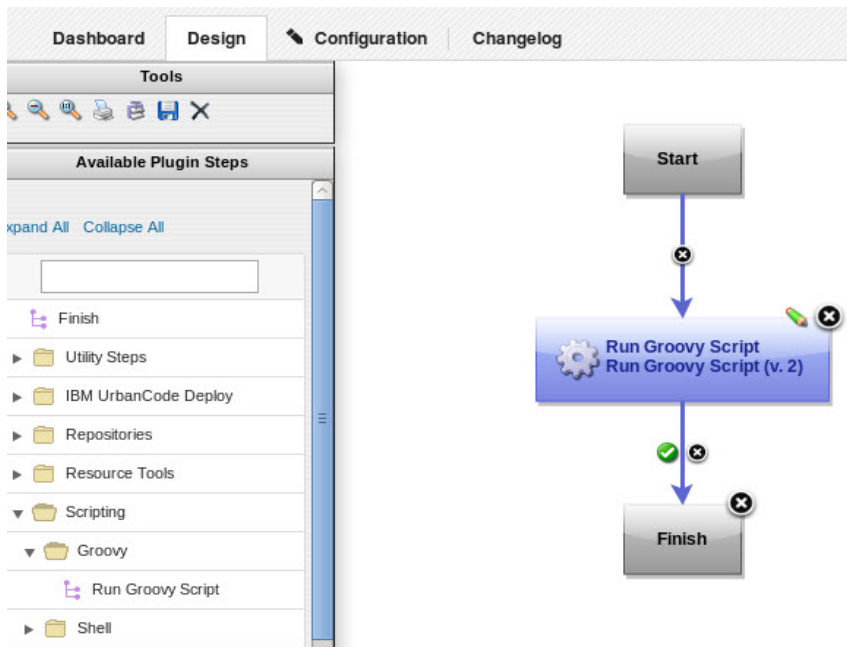
- make sure you've loaded the groovy plug-in
- set \$GROOVY_HOME to point the groovy directory under the agent.. [agent install dir]/opt/groovy-...
- add \$GROOVY_HOME/bin to path..scripts get executed by calling **groovy [scriptName] [args]**

Writing a groovy script and generic process

1. Make a generic process named "hello world groovy"
2. Make a process property called "worldName" and mark it required

The screenshot shows the 'Edit Property' dialog box in a process design tool. The dialog has a title bar with a close button. Inside, there are several fields: 'Name' with the value 'worldName', 'Description' with 'Name of world to say hello to', 'Label' with 'world name', 'Pattern' (empty), 'Required' with a checked checkbox, 'Type' with a dropdown menu showing 'Text', and 'Default Value' with 'earth'. At the bottom right are 'Save' and 'Cancel' buttons. The background shows a sidebar with tabs like 'Applications', 'Configuration', 'Processes', 'Resources', 'Calendar', 'Work Items', 'Reports', and 'Settings'. The 'Processes' tab is active, and a 'Process' list is visible on the left.

3. Add a step to the process diagram to run a groovy script



4. Add something for the script to do... In this example the process takes a value accepted from the user, creates a string and writes it to a file.

Files are pretty easy to read, write and iterate through. In this example the step writes a string to a file using the File Object

Here are docs for the file object, and a some examples

<http://groovy.codehaus.org/groovy-jdk/java/io/File.html>

<http://groovy.codehaus.org/JN2015-Files>

Running OS Commands - Operating system commands can be executed by calling the execute method on a string. ex: **"ls -la".execute()**

Here are some docs for the execute command

[http://groovy.codehaus.org/groovy-jdk/java/lang/String.html#execute\(\)](http://groovy.codehaus.org/groovy-jdk/java/lang/String.html#execute())

The screenshot shows a 'Edit Properties' dialog box for a step named 'Run Groovy Script'. The dialog has several fields and options:

- Name ***: Run Groovy Script
- Groovy Home ***: \${GROOVY_HOME}
- Script ***: A text area containing Groovy code:

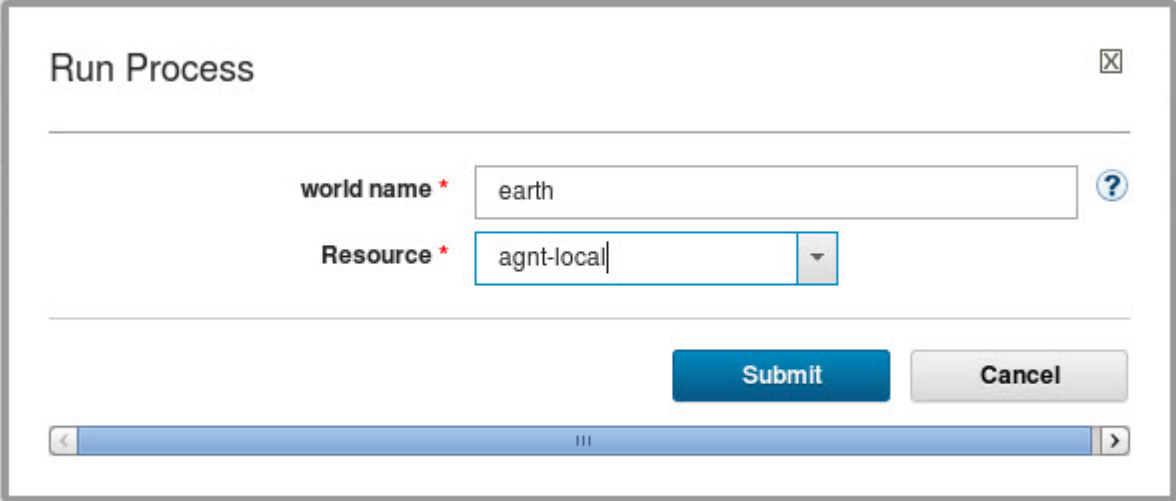
```
// capture the process property
helloStr = "hello ${p:worldName}"
println helloStr

// create a file and write to it
new File("/tmp/testFile").write(helloStr)

// do something from the command line
cmdStr = "cat /tmp/testFile"
def proc = cmdStr.execute()
proc.waitFor()
println "rtn code: ${proc.exitValue()}"
println "stdout:  ${proc.in.text}"
println "stderr:  ${proc.err.text}"
```
- Allow Failure**: A checkbox that is currently unchecked.
- Working Directory**: An empty text field.
- Post Processing Script**: A dropdown menu set to 'Step Default' and a 'New' button.
- Precondition**: An empty text field.

Each field or option has a question mark icon for help.

5. Choose a resource, enter a value and run the process. For simplicity I've kept everything local... I'm running as root on the UCD server, the script is in my home directory, and there's an agent running locally on the server as well.



The image shows a 'Run Process' dialog box with a title bar and a close button. It contains two input fields: 'world name' with the value 'earth' and a help icon, and 'Resource' with a dropdown menu showing 'agnt-local'. Below the fields are 'Submit' and 'Cancel' buttons. A progress bar is at the bottom.

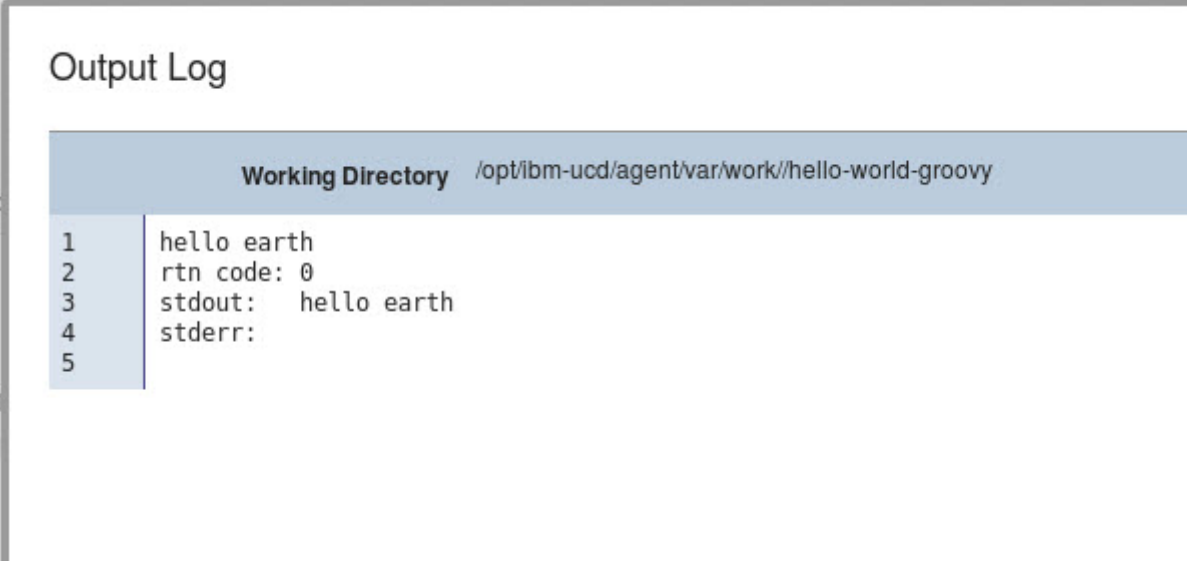
Run Process

world name * earth ?

Resource * agnt-local

Submit Cancel

6. The log shows the result of running the process...



The image shows an 'Output Log' window with a title bar. It displays the 'Working Directory' as '/opt/ibm-ucd/agent/var/work/hello-world-groovy'. Below this, a log table shows the output of the process.

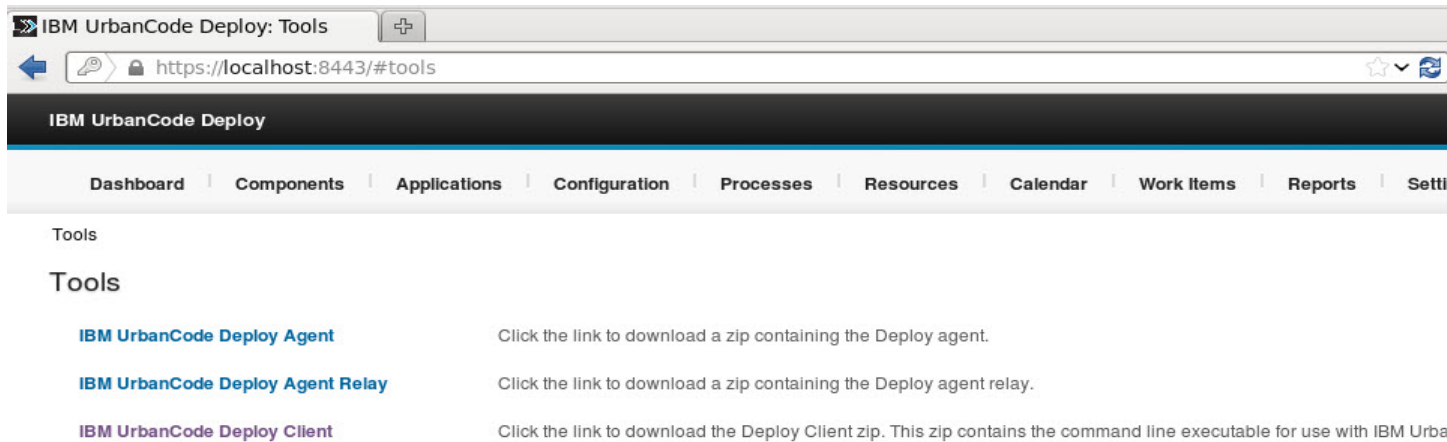
Output Log

Working Directory /opt/ibm-ucd/agent/var/work/hello-world-groovy

1	hello earth
2	rtm code: 0
3	stdout: hello earth
4	stderr:
5	

Calling the UCD Client from a groovy script

Get the UCD client from the tools page, unzip it and its ready to go.



Call it with url for the server, username and password to get the equivalent of "usage" for the client.... help is available for each of the features by adding the target command and -h to get details.

```
root@udpotexp:/opt/ibm-ucd/udclient
File Edit View Search Terminal Help
[root@udpotexp udclient]#
[root@udpotexp udclient]#
[root@udpotexp udclient]#
[root@udpotexp udclient]#
[root@udpotexp udclient]# pwd
/opt/ibm-ucd/udclient
[root@udpotexp udclient]#
[root@udpotexp udclient]# ls
udclient udclient.cmd udclient.jar
[root@udpotexp udclient]#
[root@udpotexp udclient]#
[root@udpotexp udclient]# ./udclient -weburl https://localhost:8443 -username admin -password admin
NAME
    udclient - command line client for IBM UrbanCode Deploy

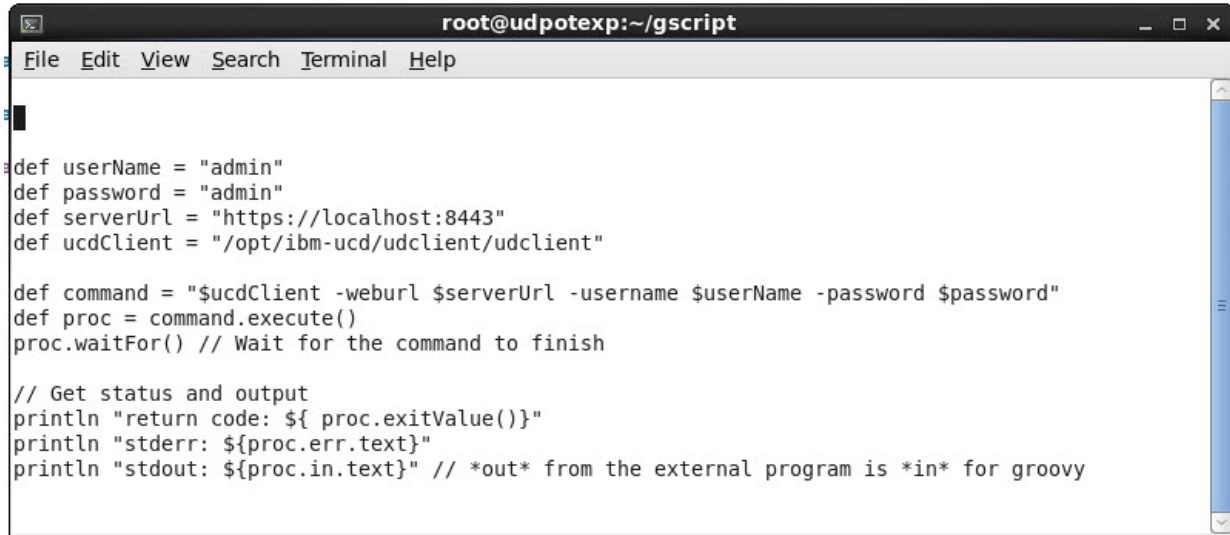
SYNOPSIS
    udclient [global-args...] [global-flags...] <command> [args...]

    command
        One of the named commands below in the Commands section
```

The generic process example above makes a call to an operating system program... calling the client from groovy is handled in a similar fashion

<http://groovy.codehaus.org/Executing+External+Processes+From+Groovy>

This example gets usage for the client, but does it from a groovy script.



```
def userName = "admin"
def password = "admin"
def serverUrl = "https://localhost:8443"
def ucdClient = "/opt/ibm-ucd/udclient/udclient"

def command = "$ucdClient -weblurl $serverUrl -username $userName -password $password"
def proc = command.execute()
proc.waitFor() // Wait for the command to finish

// Get status and output
println "return code: ${proc.exitValue()}"
println "stderr: ${proc.err.text}"
println "stdout: ${proc.in.text}" // *out* from the external program is *in* for groovy
```

Here are more details on managing processes.

<http://groovy.codehaus.org/Process+Management>

http://pleac.sourceforge.net/pleac_groovy/processmanagementetc.html

Here's an example of uploading files to a component with a groovy script



```
def String udClient(String command){
    def proc = command.execute()
    proc.waitFor()
    println "return code: ${proc.exitValue()}"
    println "stdout:      ${proc.in.text}"
    println "stderr:      ${proc.err.text}"
} // end udClient()

def userName = "admin"
def password = "admin"
def serverUrl = "https://localhost:8443"
def ucdClient = "/opt/ibm-ucd/udclient/udclient"
def component = "comp1"
def version = "ver-0080"
def workDir = "/root/gscript/tmp"

// create new version
def createVersionCmd = "$ucdClient -weblurl $serverUrl -username $userName -password $password createVersion -component $component -name $version"
println "$createVersionCmd"
udClient(createVersionCmd)

//upload the target files to the component
def addVersionFilesCmd = "$ucdClient -weblurl $serverUrl -username $userName -password $password addVersionFiles -component $component -version $version -base $workDir"
println "$addVersionFilesCmd"
udClient(addVersionFilesCmd)
```