

WARHORN: USER GUIDE

TABLE OF CONTENTS

1.0	WHAT IS WARHORN?	4
2.0	HOW DOES WARHORN WORK	5
3.0	PREREQUISITES	6
4.0	HOW TO INSTALL WARHORN	7
5.0	THE WARRIOR DIRECTORY STRUCTURE	8
5.1	Actions Directory	8
5.2	Product Drivers Directory	8
5.3	Framework Directory	9
5.4	Tools Directory	9
5.5	Warrior Core Directory	10
5.6	Warriorspace Directory	10
5.6.1	Testcases Directory	10
5.6.2	Suites Directory	10
5.6.3	Projects Directory	10
5.6.4	Data Directory	10
5.6.5	Config_files Directory	10
5.6.6	Wrapper_files Directory	10
5.6.7	Execution Directory	11
5.7	WARHORN DIRECTORY STRUCTURE	11
6.0	HOW TO CONFIGURE THE DEFAULT_CONFIG.XML	12
6.1	THE <WARHORN> TAG	12
6.2	THE <WARRIOR> TAG	13
6.3	THE <KATANA> TAG	14
6.4	THE <DRIVERS> TAG	14
6.5	THE <WARRIORSPEACE> TAG	16
7.0	HOW TO RUN WARHORN	17
8.0	LOG FILES	18
9.0	SAMPLE DEFAULT CONFIG FILE	18

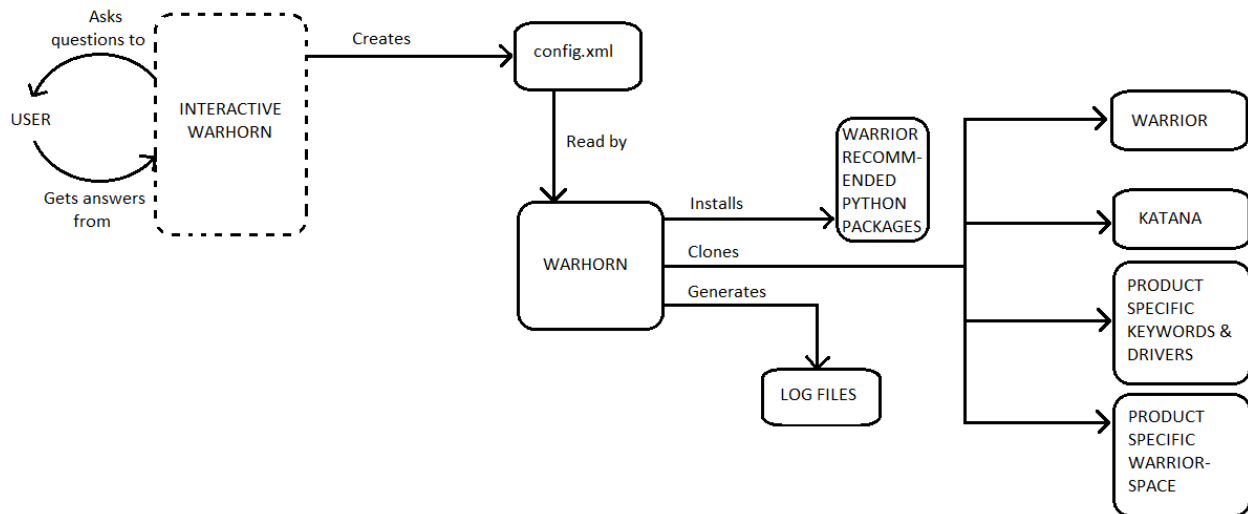
1.0 WHAT IS WARHORN?

Warhorn is Warrior's installation tool. Warhorn has the capability to install the python packages that are recommended by Warrior, Warrior, Katana – another Warrior tool, Product Specific Keywords and Drivers, and Warriorspace for you in your system.

With Warhorn, you can customize your Warrior environment to suit your needs and specifications (This is explained in more detail in section 6.0).

2.0 HOW DOES WARHORN WORK

Warhorn's premise is quite simple. Diagrammatically, it can be represented this way:



There are basically just three main components in Warhorn – the Interactive Warhorn, the Configuration file in XML format, and Warhorn.

The [Interactive Warhorn is an automated way to generate a configuration file](#). Warhorn can be run in the interactive mode by this command:

```
python warhorn.py -interactive
```

The interactive mode in the diagram above has been drawn with a dotted line – this indicates that it is not necessary to use the interactive mode to create the configuration file. You can create the configuration file (see section 6.0 for additional information) manually, in an editor - Gedit, Notepad++, or if you are feeling adventurous, vim - but the only caveat is that the configuration file has to be in a specific format as defined by Warhorn. Hence, it is recommended that you use the interactive mode.

The next component is the [configuration file](#) itself. It is essentially an [XML file that acts as a data store for Warhorn](#). All the information provided by you in the configuration file is used by Warrior to install Warrior recommended dependencies and repositories in your system.

The [third component is Warhorn](#) – it holds all the intelligence necessary for cloning and installing stuff on your system. Warhorn produces log files (section 8.0) which stores information and details about what happened during the Warhorn execution.

3.0 PREREQUISITES

Warhorn requires the following to run successfully:

1. A Linux system
2. Python – version 2.7.6+ till the latest in the 2.7 family
3. Git

Warhorn by default would install the dependencies recommended by Warrior. For this, warhorn may require you to have privileges to install packages on your system.

4.0 HOW TO INSTALL WARHORN

The Warhorn Installation Tool is located at:

```
https://github.com/warriorframework/warriorframework/tree/master/warhorn
```

If you want to clone Warhorn into a particular location, type in:

```
cd path_to_the_desired_directory
```

Now that the system has git installed on it, open the terminal and type in to clone Warhorn:

```
git clone https://github.com/warriorframework/warriorframework.git
```

The warhorn is part of warrior framework now:

```
cd warriorframework/warhorn
```

If you want to checkout a particular, branch, commit-id, or tag, type in:

```
cd warriorframework
```

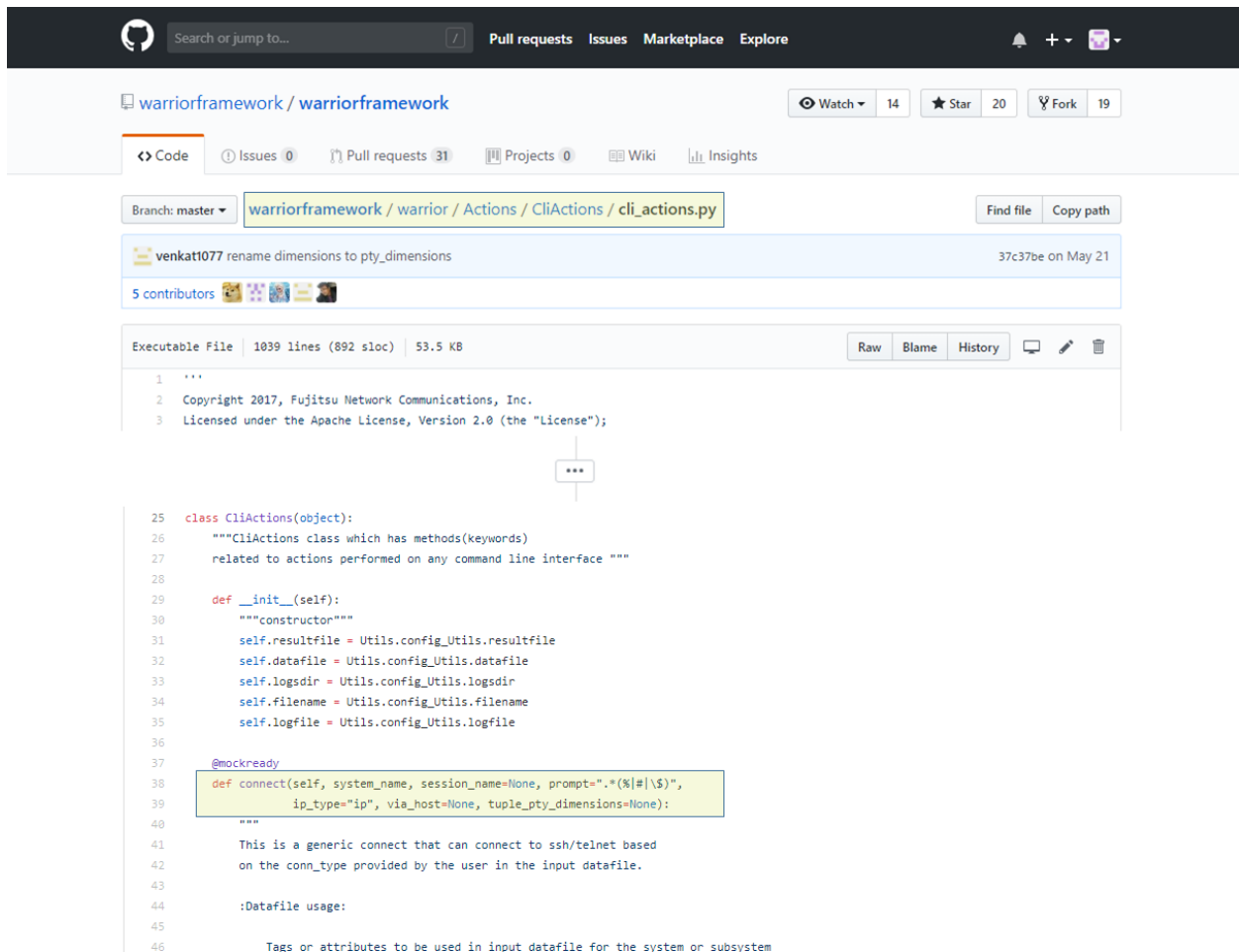
This command gets you into the newly cloned warhorn directory. Then type in:

```
git checkout branch name/tag name/commit-id
```

5.0 THE WARRIOR DIRECTORY STRUCTURE

5.1 Actions Directory

This directory contains the python libraries that contain the keyword functions. Any future keyword function development will need to be added to either an existing library or to a new library within the Actions directory. For example, the connect keyword is defined as a function in the cli_actions.py, which is inside the CliActions directory within the Actions directory.



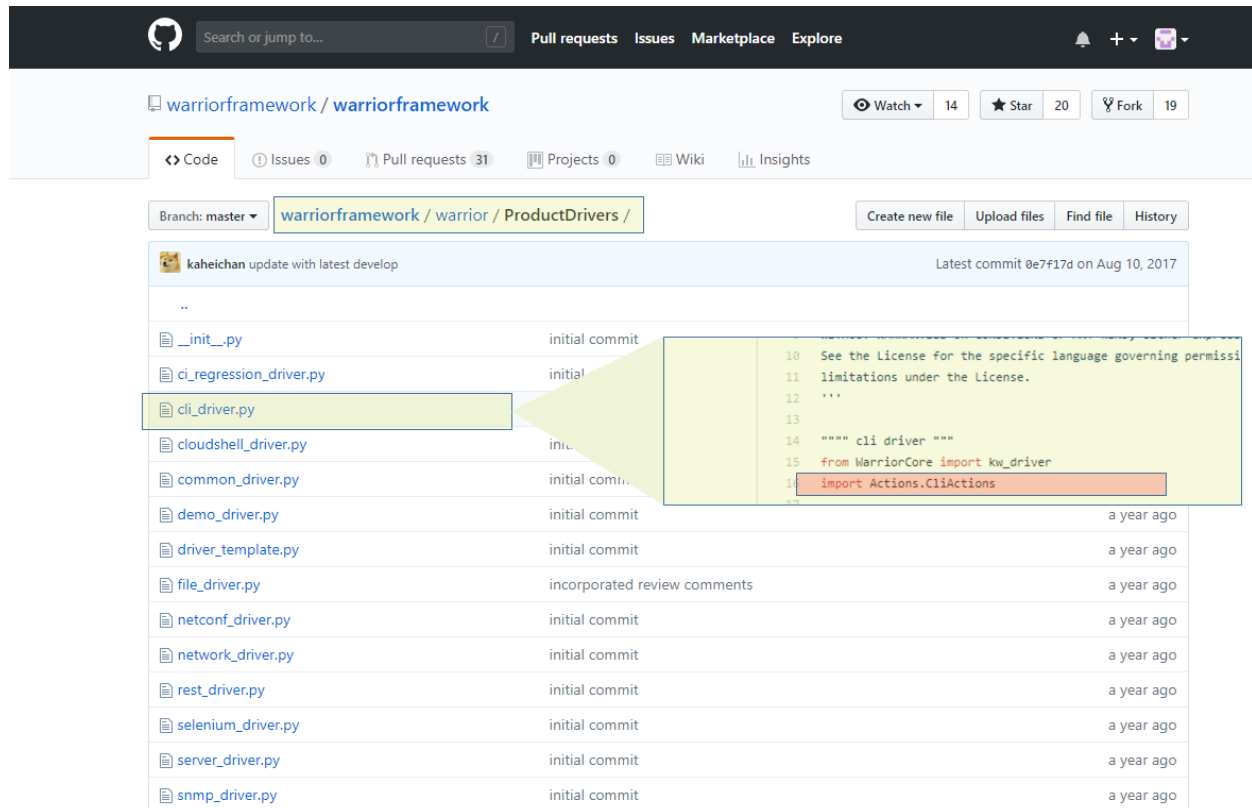
The screenshot shows the GitHub interface for the repository `warriorframework/warriorframework`. The file `cli_actions.py` is selected, showing its commit history and code. The code defines the `CliActions` class, which is a generic connect function that can connect to ssh/telnet based on the `conn_type` provided by the user in the input datafile.

```
1 '''
2 Copyright 2017, Fujitsu Network Communications, Inc.
3 Licensed under the Apache License, Version 2.0 (the "License");
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25 class CliActions(object):
26     """CliActions class which has methods(keywords)
27     related to actions performed on any command line interface """
28
29     def __init__(self):
30         """constructor"""
31         self.resultfile = Utils.config_Utils.resultfile
32         self.datafile = Utils.config_Utils.datafile
33         self.logsdir = Utils.config_Utils.logsdir
34         self.filename = Utils.config_Utils.filename
35         self.logfile = Utils.config_Utils.logfile
36
37     @mockready
38     def connect(self, system_name, session_name=None, prompt=".*(%#|\$)",
39                ip_type="ip", via_host=None, tuple_pty_dimensions=None):
40         """
41         This is a generic connect that can connect to ssh/telnet based
42         on the conn_type provided by the user in the input datafile.
43
44         :Datafile usage:
45
46         Tags or attributes to be used in input datafile for the system or subsystem
```

5.2 Product Drivers Directory

The Warrior driver files reside in the Product Drivers directory. Drivers are python files that make up the execution layer of the Warrior Framework. The product drivers will interact with the xml based testcase and start the execution of the steps based on the product driver indicated in the step. For

example, product driver for all the keywords in the cli_actions.py is cli_driver, hence it invokes execution of all keyword functions that can be used for automating a command line interface based system.



The screenshot shows the GitHub repository for `warriorframework/warriorframework`. The file list under `warrior/ProductDrivers/` includes `cli_driver.py`, which is highlighted. A code preview for `cli_driver.py` is shown on the right, containing the following code:

```
10 See the License for the specific language governing permissions
11 limitations under the License.
12 ...
13
14 """ cli driver """
15 from WarriorCore import kw_driver
16 import Actions.CliActions
```

5.3 Framework Directory

Warrior Framework's system layer is in the Framework directory. The generic framework libraries that support the functional and execution layer are located in this directory. For example, the `connection.py` has three defined functions: `connect`, `connect_ssh` and `connect_telnet` and provides the Warrior Network connectivity module. If you need functionalities like parsing the XML, parsing the JSON, diagnostics related functionality etc., this is the directory you should be looking into. You can learn more about the specifics of the libraries by reading the warrior documentation and surfing this directory.

5.4 Tools Directory

The tools directory contains directories containing files specific to Warrior Frameworks integration with other tools. This directory is not a topic of interest for this confluence page. If interested, you can learn about this from the warrior documentation.

5.5 Warrior Core Directory

This directory contains the python files that compose Warrior Framework's execution engine. Do not make any changes to the files in this directory.

5.6 Warriorspace Directory

Warrior Framework uses this directory as the default location for case, data files and execution results.

ryadaval-warrior updated documentation for testdata new options.		Latest commit 079b6e0 on Apr 3
..		
Config_files	updated documentation for testdata new options.	5 months ago
Data	Added documentation for wtag feature in Input_data_File_Template	2 years ago
Execution	initial commit	3 years ago
Projects	war-1364: update path in demo project	2 years ago
Suites	Warrior 3.0 changes	2 years ago
Testcases	adding _config.yml for website theme	2 years ago
wrapper_files	Added comments in sample wrapper file	8 months ago

5.6.1 Testcases Directory

This directory will include the <tc_case_name>.xml files which use the keywords to implement the tests to be executed. You can use a different directory to save the cases.

5.6.2 Suites Directory

This directory will include the <ts_suite_name>.xml files that define the suites to be executed.

5.6.3 Projects Directory

This directory will include the <pj_project_name>.xml files that define the projects to be executed.

5.6.4 Data Directory

The Data directory will contain the input data files as indicated in the <td_case_name>.xml. If the <case_name>.xml does not define a file, Warrior will search for data file with the same name as the <case_name>_Data.xml. If a data is not found, Warrior will assume that the case does not require a data file.

5.6.5 Config_files Directory

The Config_files directory will contain the configuration files as indicated in the input data files. Files like testdata XML files and variable config XML files will reside here.

5.6.6 Wrapper_files Directory

The wrapper_files directory will contain the test case/Suite wrapper like setup cleanup debug etc.

5.6.7 Execution Directory


Warrior stores all the execution related information like the log and result files in the Execution directory by default.

5.7 WARHORN DIRECTORY STRUCTURE

The current Warhorn directory structure looks something like this:


docs	initial commit	3 years ago
source	fix for war-1722	6 months ago
user_generated	Update config_sample.xml	last year
default_config.xml	Merge pull request #430 from warriorframework/WAR-1920	last year
readme.txt	Removed instances older repository url	2 years ago
warhorn.py	support authentication for driver, warriorspace & tools	last year

There are three folders and four files inside. The first is the Docs directory which stores the Warhorn User Guide.

docs			
Name	Date modified	Type	Size
 Warhorn User Guide	6/7/2016 1:52 PM	Adobe Acrobat D...	503 KB

The second directory – source – contains all the files that Warhorn uses to work. You should not change any contents of this directory.

Then comes the user_generated directory. This is the default directory for storing files created via the interactive mode.

user_generated			
Name	Date modified	Type	Size
 temp	6/7/2016 1:52 PM	Text Document	1 KB

The four files in the directory are – the .gitignore, which is just a file specifies intentionally untracked files that Git should ignore. The readme.txt – this file details the brief introduction to Warhorn, the default_config.xml which you are encouraged to open, read through, and edit, and finally the warhorn.py is the Warhorn executable.

6.0 HOW TO CONFIGURE THE DEFAULT_CONFIG.XML

The configuration files are an integral part of running the Warhorn tool. Warhorn, when freshly cloned, comes with a default_config.xml file with it.

This configuration file feeds data to warhorn.py and tells it what to install and clone on your machine. The default_config.xml can be edited but it is strongly recommended that it not be moved from its original location and that its name not be changed as then, the command for running Warhorn would change (see section 5.1)

Any configuration file, including the default_config.xml file, should contain these five tags - <warhorn>, <warrior>, <katana>, <drivers>, and <warriorspace>. Out of these tags, only the <warrior> tag is the mandatory one.

6.1 THE <WARHORN> TAG

The <warhorn> tag is the first tag in the configuration file.

```
▼<warhorn name="Warhorn">
```

This tag contains information about the dependencies and their version that Warhorn would install by default. The comment below notes all that information

```
▼<!--
    The required versions for each of the dependencies given below are:
        jira: 1.0.3 - Not needed for Warrior version 2.1.1 and above
        lxml: 3.5
        ncclient: 0.4.6 - Not needed for Warrior version 1.9 and above
        paramiko: 1.16.0
        pexpect: 3.1
        pysnmp: 4.3.1
        requests: 2.9.1
        selenium: 2.48.0
    Your system may or may not have the correct version installed.
    ** Set the 'install' attribute to yes if you want to install the
    dependency in your system.
    ** Set the 'correct_version' attribute to yes if you want to upgrade the
    existing package to the required version
-->
```

Since, Warhorn by default, clones the latest version of Warrior, unless specified otherwise, Jira and ncclient, have been turned “off” in the default_config.xml. So dependency section in the default_config.xml looks something like:

```

▼<!--
    Necessary for Jira module to log defects to Jira automatically
-->
<dependency name="jira" install="no"/>
<!-- Used by IronClaw tool -->
<dependency name="lxml" install="yes"/>
<!-- Used for netconf operations -->
<dependency name="ncclient" install="no"/>
<!-- Used by ncclient -->
<dependency name="paramiko" install="yes"/>
<!-- Used by cli utilities -->
<dependency name="pexpect" install="yes"/>
<!-- Used for snmp operations -->
<dependency name="pysnmp" install="yes"/>
<!-- Used for rest operations -->
<dependency name="requests" install="yes"/>
<!-- Used for web based testing -->
<dependency name="selenium" install="yes"/>

```

Here all the dependencies recommended by Warrior have their own `<dependency></dependency>` tags. The dependency name is given as value to the name attribute and whether you want to install that dependency – ‘yes’ or ‘no’ is given as value to the install attribute. The dependency corresponding to the install attribute marked as ‘yes’ or ‘no’ will only get installed if the install attribute is set to ‘yes’.

The `</warhorn>` closing tag marks the end of this section

```
</warhorn>
```

6.2 THE <WARRIOR> TAG

This tag carries with it information regarding the cloning of Warrior.

```

<warrior url="http://rtx-swtl-git.fnc.net.local/scm/war/warrior_main.git" destination=""
        label="" clean_install_warrior="">
</warrior>

```

This is a mandatory tag. So, please make sure every configuration file contains this tag and that it has been filled out with the correct information.

The attributes in the `<warrior></warrior>` tag:

Attribute	Description
url	Holds the URL of the warrior repository. The url tag has already been prepopulated in the default_config.xml. You can change that if you want to, but make sure that it is a valid url.

destination	Contains the path to the directory in which you want to clone Warrior. If it is left empty, then Warrior would be cloned in the same directory as Warhorn.
label	Indicates the branch name, tag name, or commit-id that you may want to checkout. If it is left empty, then the latest version of Warrior will be cloned.
clean_install	This tag lets you indicate whether you want to delete the existing Warrior in your system or not. If the tag value is set to 'yes', the existing Warrior will get deleted. If the tag is set to 'no', or is left empty, or is taken out altogether, existing Warrior will not get deleted.

6.3 THE <KATANA> TAG

This tag carries with it information regarding the cloning of the Warrior Tool - Katana.

```
<katana url="http://rtx-swtl-git.fnc.net.local/scm/war/katana.git"
        destination="" label="" clean_install="" clone="yes"></katana>
```

The attributes in the <katana></katana> tag:

Attribute	Description
url	Holds the URL of the Katana repository. The url tag has already been prepopulated in the default_config.xml. You can change that if you want to, but make sure that it is a valid url.
destination	Contains the path to the directory in which you want to clone Katana. If it is left empty, then Katana would be cloned in the same directory as Warhorn.
label	Indicates the branch name, tag name, or commit-id that you may want to checkout. If it is left empty, then the latest version of Katana will be cloned.
clean_install	This tag lets you indicate whether you want to delete the existing Katana in your system or not before cloning a fresh one. If the tag value is set to 'yes', the existing Katana will get deleted. If the tag is set to 'no', or is left empty, or is taken out altogether, existing Katana will not get deleted.
clone	Katana would be cloned only if this ta is set to "yes". If it is left empty, or set to "no", Katana will not be cloned.

6.4 THE <DRIVERS> TAG

Warhorn provides you with the ability to clone entire repositories containing all drivers and actions packages or you can select only the drivers that you need and the corresponding actions packages with it will be cloned for you.

This is the format for adding specific drivers that you want to clone from a particular Keyword repository:

```
<repository url="http://repository/one/url.git" clone="yes" label=""
  all_drivers="no">
  <driver name="driver_one_name" clone="yes"></driver>
    <driver name="driver_two_name" clone=""></driver>
    <driver name="driver_three_name" clone="no"></driver>
    <driver name="driver_four_name"></driver>
</repository>
```

This is the format for cloning the entire Keyword repository:

```
<repository url="ssh://username@rtx-swt1-git.fnc.net.local/scm/warkey/utp.git"
  clone="yes" label="" all_drivers="yes" username="" password="">
</repository>
```

Please provide the username and password as shown above.

Attributes in a drivers tag:

Attribute	Description
url	Holds the URL of the repository that you want to clone.
all_drivers	This attribute lets you clone all the drivers and all the actions packages when set to yes. On the chance that you do not want to clone all the drivers, but only want to clone specific drivers, the following steps need to be taken: <ol style="list-style-type: none">1. Set the all_drivers attribute to 'no'2. Create a driver tag under the repository tag as mentioned in the sample3. Fill out the attribute of that driver tag. The driver tag has one attribute – name that takes in the name of the driver that you want and another attribute "clone" that lets you turn "off" a driver cloning.4. Repeat steps 2 and 3 till all the needed driver tags are created.
label	Indicates the branch name, tag name, or commit-id that you may want to checkout.
clone	Gives you the ability to "turn off" the cloning of that particular repository. If this attribute is set explicitly to 'no', only then the repository would not be cloned. If

it is set to 'yes', or is left blank, or is removed altogether, the repository will get cloned.

You can add as many repository tags as you want and Warhorn would clone all of them for you as long as all of them carry valid information. If you do not want to clone any Keyword repositories, the main drivers tag can be left empty.

6.5 THE <WARRIORSPLACE> TAG

The warriorspace tag holds information regarding the product specific warriorspace repositories that you may want to clone. The process is similar to the drivers tag - you can add as many repository tags as you want as long as all of them carry valid information. The sample below represents the way in which the warriorspace tag should be filled out.

```
<warriorspace>
  <!-- Sample
  <repository url="http://warriorspace/repository/url.git" label="f455scd00i">
  </repository>
  -->
</warriorspace>
```

This is the format for cloning the entire warriorspace repository:

```
<repository url="http://username@rtx-swtl-git.fnc.net.local/scm/was/1finity.git"
  overwrite="" label="" clone="yes" username="" password="">
</repository>
```

Please provide the username and password as shown above.

Attribute	Description
url	Holds the URL of the repository that you want to clone.
label	Indicates the branch name, tag name, or commit-id that you may want to checkout.

7.0 HOW TO RUN WARHORN

Warhorn can be run by going the command line and running the warhorn.py file using the command:

```
python warhorn.py
```

This command supplies no arguments and therefore, Warhorn uses the default_config.xml to get the data needed to perform its tasks. It is not recommended that default_config.xml be moved from its default location and the name of the file be changed to anything else, since, the command above will not work if that is done. Warhorn would still run but only if the location of the xml file is passed as an argument.

If you need to run a particular xml file, the command should be:

```
python warhorn.py path_to_directory/file_name.xml
```

This command lets Warhorn use a particular configuration file to get its data from.

The Warhorn Interactive mode can create the configuration file in the correct XML format. Running the following command creates that XML file, and gives you the option of saving and then running that newly created file. You can also directly run the configuration file without saving it. To enter the Warhorn Interactive mode, type in this command:

```
python warhorn.py -interactive
```

8.0 LOG FILES

Log files are generated each time warhorn.py is run. Log files are not generated when the Warhorn Interactive mode is used to generate the configuration file only.

There are two kinds of log files: console_log.txt and print_log.txt. The console_log.txt logs all the console output of warhorn.py, including the print statements and errors. The print_log.txt logs just the print statements.

Both these files are stored in a time stamped directory underneath the logs folder - which gets generated when warhorn.py is run for the first time – along with the configuration file that was used by warhorn.py during that particular run.

9.0 SAMPLE DEFAULT CONFIG FILE

```
<?xml version="1.0"?>
<!-- To read full documentation for this file, please refer to user_generated/config_sample.xml -->
<data>
  <warhorn name="Warhorn">
    <dependency name="jira" user="no" install="no"/>
    <dependency name="txml" user="no" install="yes"/>
    <dependency name="ncclient" user="no" install="no"/>
    <dependency name="paramiko" user="no" install="yes"/>
    <dependency name="pexpect" user="no" install="yes"/>
    <dependency name="pysnmp" user="no" install="yes"/>
    <dependency name="requests" user="no" install="yes"/>
    <dependency name="selenium" user="no" install="yes"/>
    <dependency name="xlrd" user="no" install="no"/>
    <dependency name="cloudshell-automation-api" user="no" install="yes"/>
    <dependency name="pycryptodome" user="no" install="yes"/>
  </warhorn>
  <!-- Fill this out with the details of the virtual environment to be used. The name is mandatory for the virtual environment to be used. Please note that if this is not used, then the warhorn expects the dependency packages to be already installed or admin privileges available for installing them. location is the place where virtualenv binary can be found. The virtual env would be created in the local directory itself with the name provided. If install is set to yes, then warhorn expects atleast the admin privileges to install virtual env. -->
  <virtualenv name="war_virtualenv" install="no" location="~/local/bin/virtualenv"/>
  <warriorframework clone="no" clean_install="yes" label="" url="https://github.com/warriorframework/warriorframework.git"> </warriorframework>
  <drivers>
    <!-- Fill this out with the details of the repository that you want to clone <repository url="http://repository/one/url.git" clone="yes" label="tag-name" all_drivers="no" username="" password=""> <driver name="driver_one_name" clone="yes"/> <driver name="driver_two_name" clone=""/> <driver name="driver_three_name" clone="no"/> <driver name="driver_four_name"/> </repository> -->
    <repository clone="yes" label="" url="http://VKOTAKON@rtx-swlt-git.fnc.net.local/scm/warkey/utp.git" password="" username="" all_drivers="yes"> </repository>
    <repository clone="yes" label="" url="http://VKOTAKON@rtx-swlt-git.fnc.net.local/scm/was/1finity.git" password="" username="" all_drivers="yes"> </repository>
    <repository clone="yes" label="" url="http://VKOTAKON@rtx-swlt-git.fnc.net.local/scm/warkey/kw_tseries.git" password="" username="" all_drivers="yes"> </repository>
    <repository clone="yes" label="" url="http://VKOTAKON@rtx-swlt-git.fnc.net.local/scm/warkey/kw_oleaf.git" password="" username="" all_drivers="yes"> </repository>
    <repository clone="yes" label="" url="http://VKOTAKON@rtx-swlt-git.fnc.net.local/scm/warkey/testset.git" password="" username="" all_drivers="yes"> </repository>
  </drivers>
  <warriorspace>
    <!-- Fill this out with the details of the repository that you want to clone <repository url="http://warriorspace/repository/url.git" overwrite="no" label="commit-id" clone="yes" username="" password=""> </repository> -->
    <repository clone="yes" label="" url="http://VKOTAKON@rtx-swlt-git.fnc.net.local/scm/was/1finity.git" password="" username="" overwrite=""> </repository>
    <repository clone="yes" label="" url="http://VKOTAKON@rtx-swlt-git.fnc.net.local/scm/was/cia.git" password="" username="" overwrite=""> </repository>
    <repository clone="yes" label="" url="http://VKOTAKON@rtx-swlt-git.fnc.net.local/scm/was/common.git" password="" username="" overwrite=""> </repository>
    <repository clone="yes" label="" url="http://VKOTAKON@rtx-swlt-git.fnc.net.local/scm/was/utp.git" password="" username="" overwrite=""> </repository>
  </warriorspace>
</data>
```



default_config.xml

