R • I • T	Rochester Institute of Technology Golisano College of Computing and Information Sciences Department of Information Sciences and Technologies

4002-XXX
Software Development on Linux Systems
Lab 06 – Configuration files and installing/using version control software

Name: ______________________________________ Section: _____________

Activity – Configuration Files

As we have discussed, everything in Linux is a file. The system itself is made up of many types of files, such as binaries, scripts and configuration files. Binaries are the compiled source code for a program (IE: .class files in Java or .exe in C#). Binaries are the executable code for a program. In Linux these often end in .bin or .run, but most commonly will not have any extension. Configuration files are used by the binaries or the system scripts as parameters for settings and preferences. These may be system settings, application settings or personal settings.

Configuration files come in many forms depending on the purpose, the application requiring it and the author of the configuration file. Some will be just numbers and hex codes. Some will look more like variables, and others may be comma delimited parameters. Configuration files really can take any form. Configuration files will mostly be seen without file extensions, with the .ini extension or with the .conf extension. It is expected you will probably write configuration files in this course for your own work.

You will work with configuration files in this lab.

Personal Configuration Files

In some cases, configuration files are actually scripts as well. In Linux you have several configuration files linked to your account. Some of these are scripts that run.

Your personal configuration files are in your home directory. Recall ~ means your home folder in Linux.

Note: Files beginning with a period are hidden files and are not displayed unless the user specifies they want to see hidden files.

1) First look at your user profile. This file is named **.profile** in your home directory
   Note: If .bash_profile does not exist, it will be called .profile instead
   In terminal type "cat ~/.bash_profile" and hit enter
   This will display a script that sets your default path for commands and applications when you login. This is also where you can set up certain scripts, commands and applications to run when you login. Your profile allows you configure your personal session. You may add custom configurations for your user here.

2) Now look at your bash initialization file. This is called **.**bashrc
In terminal type “cat ~/**.**bashrc” and hit enter
This should display a bash script that has Bash scripting configuration information, such as user settings, keyword highlighting preferences, aliases to other commands, how many commands to store in history, etc.

3) The previous two configuration files were bash scripts, but configuration files may take any form as long as the application knows how the parse the information. For example, a personal file you have is your bash history. This is a configuration file that contains a large list of your previous commands. This is used by bash so that when you hit the up arrow in terminal it knows which previous command to show.

Look at your own bash history. This file is called **.**bash_history
In terminal type “cat ~/**.**bash_history”
You will see your previously typed commands one per line. This file contains no other data or logic than just a line for line list of your commands.

4) To see how these files work, we will edit your profile.

Using the editor of your choice (preferably gedit or vi [vim]) edit your **.profile** file in your home directory to echo “Welcome $USER” at the very end.

Save the file and open a brand **new** terminal session. What happened?

____

Show the results of all of these to your instructor/TA to get credit

Instructor/TA Sign-Off ______________________________

System Configuration Files

Now we will look at system configuration files. System configuration files are similar to personal configuration files and may take any form, but they are used for system settings and preferences; not a specific user.

1) One you will probably be familiar with is your hosts file
Type “cat /etc/hosts”
This should show your localhost and saved host information.
It may look like this, but will vary from computer to computer

```
127.0.0.1    localhost
127.0.1.1    hostname.rochester.rr.com  hostname

# The following lines are desirable for IPv6 capable hosts
::1     ip6-localhost ip6-loopback
 fe00::0 ip6-localnet
 ff00::0 ip6-mcastprefix
 ff02::1 ip6-allnodes
 ff02::2 ip6-allrouters
```

As you can see, this file is not a Bash script, it is a just a space delimited file which an expected format for each field. The system understands this format and parses the file to set up the network. This is the file that is responsible for making the IP address 127.0.0.1 able to be used as “localhost” instead.

2) Now we will look at the file system configuration file. This file is called fstab (File system table). This file is read by the operating system to determine how to mount file systems on your hard drive(s).

In terminal type “cat /etc/fstab” and hit enter
Your output may look like this. It will vary greatly from machine to machine

```
# /etc/fstab: static file system information.
#
# Use 'blkid -o value -s UUID' to print the universally unique identifier
# for a device; this may be used with UUID= as a more robust way to name
# devices that works even if disks are added and removed. See fstab(5).
#
# <file system> <mount point>   <type>  <options>       <dump>  <pass>
proc            /proc           proc    nodev,noexec,nosuid 0       0
/dev/sda1       /               ext4    errors=remount-ro 0       1
# swap was on /dev/sda5 during installation
UUID=9550f521-8c15-4a99-8c24-0c3c6eaa9bf6 none            swap    sw              0       0
/dev/sda2       /boot           ext4    errors=remount-ro 0       1
```

You are not expected to know how to read this yet, but the fstab file is a file you can edit for custom file system mappings and file system options. It is a tab delimited file with several parameters the system reads to set up the file access.

3) Next we will look at a simple configuration file that just stores numerical data. This is the ctrl-alt-del file. This file determines the action of the keystroke ctrl + alt + delete.

In terminal type “cat /proc/sys/kernel/ctrl-alt-del” and hit enter
This will probably print a **0** or a **1** .

If the value is zero this means that if ctrl alt & delete are pushed, the system will send a request to the user to reboot.

If the value is one or above this is referred to as a hard reboot. This means if ctrl alt & delete are pushed, the system will instantly cut power and turn back on.

4) Another numerical file that you can edit (**but with extreme care**) is threads-max
Threads-max is a configuration file that stores the maximum number of threads your system can run. This file is automatically set to a safe amount for your processor, but may be edited for server performance, development use or hardware testing.

In terminal type “cat /proc/sys/kernel/threads-max” and hit enter
This will output a number, but that number will vary from system to system as it is based on the processor, RAM and swap.

Only edit this file if you have a direct reason to. By default, it will be set to a reasonable amount.

5) Now we will look at a configuration file the system uses, but you as a user are unable to change. This a file used by the system to configure software and instructions for the processor. This file is automatically generated and maintained by the system. This file is called cpuinfo.

In terminal type “cat /proc/cpuinfo” and hit enter
This will display information about each core of the processor.
Each core will look something like this. (This will vary from machine to machine)

```
processor         : 0
vendor_id         : GenuineIntel
cpu family        : 6
model             : 37
model name        : Intel(R) Core(TM) i5 CPU       M 430  @ 2.27GHz
stepping          : 2
cpu MHz           : 1199.000
cache size        : 3072 KB
physical id       : 0
siblings          : 4
core id           : 2
cpu cores         : 2
apicid            : 5
initial apicid    : 5
fpu               : yes
fpu_exception     : yes
cpuid level       : 11
cache_alignment   : 64
address sizes     : 36 bits physical, 48 bits virtual
power management:
```

Practice

In this area of the lab you will create your own fake configuration files. You are free to design them how you want, but they must be delimited parameters or scripts.

1) In your home directory create a file called “myapp.conf”

Fill this file with data so that an application could read it and recognize each field. (You may use delimiters, such as one item per line, commas, colons, tabs, etc). The idea is a fake server would read this file and then set itself up to reflect the parameters in the configuration file.

The data in this file will be fake, just make it look like a real configuration file.

The configuration file must

a) have comments starting with # that explain fields
b) store a parameter that is the maximum number of users on an imaginary server
c) store a parameter that tells how long the imaginary server has been running
d) stores a fake hostname parameter of the imaginary server
e) stores a parameter of which port to listen on
f) stores a parameter that tells whether or not the server is encrypted or not
g) stores a parameter that is the folder path for the server

2) In your home directory you will create a file called “myapp.ini”

This configuration file will be a Bash script that configures the fake server.

The script must

a) create a file called fake_server.log
b) write the date to the server log with

```
echo “server started at `date` “ >> fake_server.log;
# >> means append to file
```

c) create a file called server_on
d) check if the current minute of the hour is over 30

(hint “date +%M ” will give the number of minutes past the hour)

I) If the current minute over 30 create a file called fake_server.lock
II) If the current minute is 30 or less append a message to fake_server.log saying that the server was never locked and gives the date

e) creates a variable called SERVER_THREADS that equals 1000 **and** exports it.

Show the results of all of these to your instructor/TA to get credit

Instructor/TA Sign-Off ______________________________

R • I • T	Rochester Institute of Technology Golisano College of Computing and Information Sciences Department of Information Sciences and Technologies

Activity – Version Control Software

Version control software is an application that allows you to update files, while maintaining copies of the previous files. In the case of a problem with the current file, you can immediately revert to a previous copy of the file to temporarily allow you to continue your process. In most cases the newest version of the file is stored with the highest version number, while previous files are stored with lower version numbers. This also allows developers to track which version of a file a problem started to occur in.

Version control can be used for basically any type of file. Source code is probably the prominent occurrence, but documentation, graphics and other file types are common. In most cases all of the files an application requires are stored in version control. This means that in theory a person could grab a specific version from the version control software and would have the complete application as it was in that version including documentation, media files, configuration files, etc

Though CVS is an immensely popular version control system, it is very dated and is mostly used due to age and maturity. CVS will not be covered in this lab as it has been replaced by many advanced version control systems.

The current most popular version control systems are all open source. These are:

Name	License	Description
Subversion (SVN) *Built in: C*	Apache	Most popular version control; mature, but somewhat dated. SVN is a centralized system that code is checked in/out of. Fast and stable, but suffers from a single point of failure and clutter.
Git *Built in: C, Bash*	GPL	Git is a new decentralized version control system that has become very popular. Git is fast and efficient. With Git each user has their own copy of the repository and merges their code back into someone else's repository. Git can be confusing to use at first due to the idea of storing local repositories.
Mercurial (Hg) *Built in: Python*	GPL	Mercurial is another new decentralized version control system that has become popular. Mercurial has a lot of IDE support and has an integrated secure web service for sharing code and exploring through a browser. Mercurial is simple to use, but does not truly have local repositories.
Bazaar (Bzr) *Built in: Python*	GPL	Bazaar is designed to be an easy to use/learn human friendly version control system. It is simple to use and can be a centralized system, a decentralized system or a number of variant styles. Bazaar is not as fast as Mercurial or Git, but has a lot of flexibility. It is easy to modify and can be embedded.

Assignment

For this section of the lab you will have to install two version control systems. You will be using Bazaar and Git. Your lab write up and blog will reflect your.

Activity – Bazaar

1) Installing Bazaar

Using your preferred method of installation install Bazaar
These are the package names-

(These may already be installed on your system)

Ubuntu:	bzr
	bzrtools
	qbzr
	bzr-explorer
	openssh-server
Fedora:	bzr
	bzrtools
	qbzr
	bzr-explorer
	openssh-server

2) Checking Out code

a) create a folder in your home directory called do-repo
We will use this folder to checkout an open source project called “Do”

b) cd to the do-repo folder

c) In terminal type “bzr init”

d) Type “ bzr branch http://bazaar.launchpad.net/~do-core/do/trunk do-trunk/ ”
This creates a local repository of the code in do-trunk

3) Configuring your account

a) Type “bzr whoami 'firstname lastname <me@rit.edu>' ”
Replace the email with your RIT email address

b) Now type “bzr whoami”

4) Creating a new project

a) cd to your home directory create a new folder called “test-bzr”

b) Type "bzr init-repo test-bzr"
c) cd to the new directory test-bzr
d) Type "bzr init trunk"
e) cd to "trunk"
f) Create files called "text1.txt", "text2.txt", "text1.c" and "text2.c"
g) Now type "bzr add text1.txt text2.txt"
 This will add text1.txt and text2.txt to the commit list
h) Now type "bzr add *.c"
 This will add any file that end in extension .c
i) To commit type 'bzr commit -m "Added new files to the project" '
 This will commit the files that you added to bzr with the add command
 -m means message. The message will be stored with your commit
 Your files will now be marked as revision 1

5) GTK Explorer
a) Create a new folder in your home folder called gwibber-repo

b) Now that you have created a project from the command line type
"bzr explorer"

This is the GUI for Bazaar. As you can see you can checkout code and create new projects from here.

c) Use "get source from elsewhere" to create a new "branch"
-You will store this branch gwibber-repo.
-Unless specified use defaults
-**If asked to initialize a directory outside of a shared repository, choose yes and hit "ok" . Then hit "close"**

From: http://bazaar.launchpad.net/~gwibber-committers/gwibber/trunk/
To: /home/lastname/gwibber-repo/

After you checkout, hit close and you will be given an explorer view of the code. You do **not** need to know how to use the explorer at this point.

Note: You may also use the explorer to create a project

6) Web Viewer
a) Using your preferred method of installation install Loggerhead

These are the package names-

Ubuntu: loggerhead
Fedora: loggerhead

b) Assuming you have committed files in step 4
cd to your home folder and type " serve-branches - -allow-writes test-bzr/ "
This will start the web server and will host only that path. It will default to your IP and port 8080. If you remove the -- allow writes flag, it will become read only

[--]
This part is not required. It is just information in case you want to run your own bzr server.

If you would rather choose a specific address or port you can supply them with
"serve-branches - -host=255.255.255.255 - -port=8081 /files"
where 255.255.255.255 is your IP

or

"serve-branches - -host=www.you.com - -port=12345 /files"
where www.you.com is your website address
[--]

c) Now type "ifconfig" and determine your IP address
Note: you will probably have to open a new terminal window as if you kill serve-branches process your web server will go down.

d) Then open a browser and type in your IP address on port 8080
Example: http://192.168.1.140:8080

e) Now a different machine running Bzr can pull it with
bzr branch "http://192.168.1.140:8080/trunk /local-repo

Note: This is not sufficient security. It can be used to share code on a local network, but does not provide you much security as it is not encrypted and would be vulnerable to cross-site scripting. Though it does not necessarily let anyone into your computer, it can allow people to do cross-site scripting to force you to go to their website If you are using this for the project, be sure to end the process when you are no longer using it. If you do not want to use this, you may share your code through sftp, ssh or Apache.

7) Using Bazaar

a) On the second machine go into your new /local-repo folder that you just pulled from the web viewer and add several lines of words to "text1.c"

b) In your /local-repo directory type "bzr add text1.c" to add the changed file to your commit list.

c) Now type "bzr commit -m "Added new functionality to text1.c"

d) Then type "bzr push http://192.168.1.140:8080/trunk "
where the IP address is the IP address of the serving machine.

This will add your change back into the pulled repository, but will not update for any current users on that machine.

e) On the machine hosting the web service, open text1.c and add a couple lines.
This file will be out of date in comparison to the file you committed on the other machine.

f) Then on that same machine type "bzr add text1.c" in the folder that has the file "text1.c"

g) Then type "bzr commit -m "Added new functions"
This will add your file to the repository that is hosted, but it will not line up with the one that was committed from the other machine

8) Resolving Conflicts

In the previous section to submitted code from two different machines to one branch meaning they are out of date.

To resolve this on your second machine (NOT the one hosting the website)

a) open a web browser and look at the revision history in loggerhead. This will show you the changes between versions.

b) cd to the directory that has /local-repo in it

b) type "bzr merge"
This will grab all of the new code and add it into the files you have in the appropriate lines. Merge is how you merge updated code with yours.

c) This is where you would resolve conflicts if you needed to. For now just add a new line to the “text1.c” file

d) Add the file bzr and commit it with message “merged changes”

e) Now type “bzr push http://192.168.1.140:8080/trunk “ to push your code back with issues resolved. **The IP address is the IP address of the machine hosting the site**

f) Now look at loggerhead and you will see the merged symbol on the newest revision number. You may have to refresh.

g) The problem is this updates the hosted repository on the web server, but it does not update the working branch (the actual code directory on the web server you are committing from). It does not update because someone could be working in those files during the time of commit, as it is your repository.

To resolve this in places you have pulled from you use the “merge” command used previously.

Merge works when you are pulling from somewhere, but since you are not pulling on the web server (as that is where the code exists), you will have to do update.

In your bzr directory on the web server type “bzr update”

This will update your working branch to what is being hosted. The hosted code is the tree (everyones' committed branches). When you pull it updates your branch to the tree.

h) There may be instances where people pulling from you just want to overwrite any code they have with code from the existing tree on the server.

This is where **- -**overwrite comes in. Overwrite pulls a copy of the newest revision from the tree onto your machine and overwrites any current files you have in your branch.

On your second machine, **Not the web server** type
“bzr pull **- -**overwrite”

9) Eclipse Integration
If you want to use bzr for your project, but do not want to use Eclipse check here
http://wiki.bazaar.canonical.com/IDEIntegration

a) Open Eclipse

b) Choose a location for the workspace

c) In Eclipse, click help from the menu and choose "install new software"

d) In the "work with" field type
http://development.bitwig.com/qbzr-eclipse

e) Click "add"

f) For the name just put "Bazaar plugin" and hit "ok"

g) In the selection area where it says "uncategorized" put a check in the checkbox

h) Hit "next"

i) Hit "next"

j) Accept the license if you agree to it

k) Hit Finish

l) Accept unsigned content

m) Restart Eclipse

10) Bazaar Eclipse plugin
a) Using the method of your choice install the c/c++ tools for Eclipse
Close Eclipse first if it is open
These are the package names:

Ubuntu: eclipse-cdt
Fedora: eclipse-cdt

b) Open Eclipse

c) Go to File → New → C++ Project
d) Create a new project in your workspace called "test"
e) On the top bar go to Bazaar → Start → Checkout
f) In the branch source put http://192.168.1.140:8080/trunk
Where the IP is the IP of your web server
g) For the local directory choose your workspace and the folder called "test" for the project you just created.
h) Hit "ok"
i) Now to go window → open perspective → other → C/C++
j) On the left in the project explorer, open your test folder and you should see the freshly checked out code.
k) Open text2.c and type a few random lines; then save the file
l) In the project explorer on the left, right click on the file "text2.c" and choose Bazaar → Commit Content.
m) Add the message "Added new content to text2.c"
n) Hit "ok"

o) To update your files to the newest version you may use "merge" or "pull new revisions"
Right click on the folder "test" from the project explorer on the left and choose Bazaar → Collaborate → Pull New Revisions
p) Put in the repository URL if it does not remember it
q) Hit "ok" This will update your repository to the newest.

r) With the same process do a "merge" from the server.

R • I • T	Rochester Institute of Technology Golisano College of Computing and Information Sciences Department of Information Sciences and Technologies

Activity – Git

1) Installing Git

Using your preferred method of installation install Git

These are the package names-

(These may already be installed on your system)

Ubuntu:	git-core
	git-gui
	openssh-server
Fedora:	git-core
	git-gui
	openssh-server

2) Checking Out code

a) create a folder in your home directory called nodejs
We will use this folder to checkout an open source project called "node.js"

b) cd to the nodejs folder

c) In terminal type "git init" and hit enter
This informs git that this folder will be a repository and creates an index.

d) Type "git clone git://github.com/joyent/node.git "
This creates a local repository of the code for node.js

3) Configuring your account

a) Type "git config --global user.name "firstname lastname"

b) Type "git config --global user.email "me@rit.edu"
Replace the email with your RIT email address

4) Creating a new project
- a) cd to your home directory and create a new folder called "test-git"
- b) cd to the new directory test-git
- c) Create a new folder called "myFolder"
- c) cd to the new folder myFolder
- d) Type "git init"
- e) Create files called "text1.txt", "text2.txt", "text1.c" and "test2.c"
- f) Now type "git add text1.txt text2.txt'
- g) Now type "git add *.c "
  This will add any files that end in extension .c
- h) To commit this type "git commit -m "Added new files to the project"
  This will commit the files that you added to git with the add command.
  -m means message. The message will be stored with your commit.
  Your files will not be marked as revision 1

5) Gitk
- a) Create a new folder in your home folder called jquery-repo
- b) Now that you have created a folder cd to that directory
- c) Type "git clone git://github.com/jquery/jquery.git "
- d) Type "gitk"

  This is the GUI for Git. This does not provide any pull or clone support, but it does allow you to merge remote code with existing code and visualize your repository.

  The tool also allows you to commit and push.

6) Web Viewer
- a) As the complete options for a Git web viewer require Apache or third part downloads we will skip these section for the sake of this lab.

7) Using Git

a) Run the command "service ssh start" on your machine. You will need to run this as a root user. This will start ssh for transferring across machines.

b) On a second machine create a folder called local-repo in your home folder

c) cd to the new local-repo folder

d)On the second machine type
"git clone *lastname@ip.add.re.ss*:/home/*lastname*/test-git/myFolder/

where *lastname* is your username on the FIRST machine and *ip.add.re.ss* is the IP address of your FIRST machine.

e) Now you will see the myFolder repo you pulled from your other machine. Open the folder and open the file "text1.txt". Add several lines of sentences to the file and save it.

e) In the local-repo directory type "git add text1.txt" to add the changed file to your commit list.

f) Commit these changes with "git commit -m "Modified text1.txt"

8) Resolving Conflicts

a) On your SECOND machine install openssh-server as you did earlier in this lab

b) On your FIRST machine cd into the myFolder directory

b) Now on your first machine add the remote client with
" git remote add machine2 *lastname@ip.add.re.ss*:/home/local-repo/myFolder/

where *lastname* is your username on the FIRST machine and *ip.add.re.ss* is the IP address of your FIRST machine.

d) Fetch the updated code from that machine with "git fetch machine2 "

This will add your change back into the repository, but will not update for the user on that machine.

e) On the machine hosting the web service (machine1) , open text1.txt and add a couple lines.
This file will be out of date in comparison to the file you committed on the other machine.

f) Then on that same machine type “git add text1.txt” in the folder that has the file “text1.txt”

g) Then type “git commit -m “Added new functions”
This will add your file to the repository that is hosted, but it will not line up with the one that was committed from the other machine

h) Now merge the changes from both machines. On the first machine type
“git merge machine2/master”

This will warn you about conflicts and add the conflict info into your file

i) On machine, open the text1.txt file. You will see arrows pointing in what is new from the master branch (head) and machine2 and equals signs between the code. Remove these arrows and the equals signs to merge the code. This is where you would resolve code conflicts if there were any by removing the unnecessary lines. Save the file.

j) Now add the new file back into the branch with
“git add * ”

k) Commit the new merged file back in with
“git commit -m 'Merged changes from machine2'

l) Now on machine2 get the latest code with
“git fetch origin”

m) Now merge the updated code back into machine2's branch. This is referred to as fast-forward.
“git merge origin/master”

9) Eclipse Integration
   a) Open machine2 Eclipse

   b) Choose a location for the workspace

   c) In Eclipse, click help from the menu and choose “install new software”

   d) In the “work with” field type
      http://download.eclipse.org/egit/updates/

   e) Click “add”

   f) For the name just put “Git plugin” and hit “ok”

   g) In the selection area where it says “Eclipse Git Team Provider” and “Jgit”
      Expand the “Eclipse Git Team Provider” section with the arrow next to it.

   h) Put a checkmark “Eclipse EGit” option

   i) Hit “next”

   j) Hit “next”

   k) Accept the license if you agree to it

   l) Hit Finish

   m) Restart Eclipse

10)Bazaar Eclipse plugin
   a) Using the method of your choice install the c/c++ tools for Eclipse
      Close Eclipse first if it is open
      These are the package names:

      Ubuntu: eclipse-cdt
      Fedora: eclipse-cdt

   b) Open Eclipse
   c) On the top bar go to Window → Open Perspective → Other
   d) Choose “Git Repository Exploring” and hit “ok”

e) Click “Clone a git repository”
f) For the host put in the ip address of machine1
g) For the repo path type “ /home/*lastname/*test-git/myFolder/ “

where *lastname* is the username of machine1

h) For the Protocol choose “ssh” and port choose “22”
i) For the user type in the username of your user on machine1
j) For the password type in the password of the user on machine1 and hit “next”
k) Choose “yes” for the SSH RSA key
l) Make sure “master” is marked in the branch selection and hit “next”
m) Hit “Finish”
n) Right click on the new myFolder repo listed under “Git Repositories” in the Git Repositories view.
o) Choose “Import Projects”
p) At the import projects window choose “Use the New Projects Wizard” and click “finish”
q) At the new project screen choose “C++ Project” under “C/C++”
r) For the name type in “git-test” and hit “Finish”
s) Say “No” when it asks to switch perspectives
t) On the left in the Git Repositories window, open the test2.c file. It should show on the right after opened. Add a couple lines to the file and save it.
u) Right click on the myFolder repository and choose “commit”
v) Enter a commit message and hit “commit”

11) Merging changes back in

a) Eclipse created a directory on machine2 at /home/*lastname*/test-git/myFolder/
You need to pull these changes into machine1 now.

b) On machine1 in the test-git/myFolder directory type
“git remote add eclipseMachine *lastname@ip.add.re.ss*:/home/test-git/myFolder/ ”

c) Now type “git fetch eclipseMachine”
This will retrieve the changes from the machine with Eclipse on it.

d) Now merge the changes in with “git merge eclipseMachine/master ”

This will have merged all of the changes you made in eclipse back in.