GIT (and Version Control Systems)

Contents

1 Skills/Concepts **1**

1.1 Version Control Systems . . . 2

1.2 Console Use . . . 2

1.3 Getting Help for Commands . . . 2

1.4 Manipulating Files . . . 2

2 Preparation **2**

3 Create a GIT Repository **3**

3.1 Background . . . 3

3.2 Process . . . 3

3.3 Conclusion . . . 3

4 Add and Modify a File **3**

4.1 Background . . . 3

4.2 Process . . . 4

4.3 Conclusion . . . 5

5 Delete a File (and get it back) **5**

5.1 Background . . . 5

5.2 Process . . . 5

5.3 Conclusion . . . 5

1 Skills/Concepts

This module will include the following (among other) concepts:

- version control systems
- console use
- filesystems
- manipulating files

1.1 Version Control Systems

Version control systems provide a means of tracking changes to files over time. That is, they allow retrieving past versions of a file. Generally, the developer needs to designate checkpoints and choose to save a given version of a file. Then, as needed, special commands may be used to fetch previous versions, compare current with previous versions, compare previous versions with other previous versions, or perform a wide array of related tasks. Generally there is some other metadata stored with previous file revisions, including date, author, and optional comments.

Some other version control systems (which will not be covered):

- CVS
- Mercurial
- Subversion

1.2 Console Use

This module relies heavily on the command-line. You will be using a terminal emulator to execute the following commands.

Command	Description
git	A version control system
rm	Remove files
ls	Lists files
man	View on-line manual pages

1.3 Getting Help for Commands

We are just touching on some features of git and wont be using it, but the `git` command has a whole host of on-line help that can be viewed via either `git help` or via the `man` command.

In addition, when doing these operations on your own, the following are excellent resources for help with Git:

- Git-SCM Web Site which provides an excellent book called Pro Git
- Stack Overflow

1.4 Manipulating Files

You'll be using an editor (your choice) to edit file content, and then you'll be using the command line to create, delete, and restore files.

When we use a terminal emulator, we will be using the Bash shell. This shell, like many others, supports redirection. We will be using the echo command in conjunction with output redirection when we create files.

The specific redirection operator we will use is the standard out redirection operator >. The command to the left of the > will be redirected to the file named on the right of the >.

For example:

```
echo hello > /tmp/hello.txt
```

This example invokes echo, which just repeats on standard out whatever text it is given, and saves the result of echoing `hello` to the file `/tmp/hello.txt`. That is, after running the above command, in the `/tmp/` directory, there will be a file named `hello.txt` that contains the text `hello`.

2 Preparation

You should be fine booting up any Linux distribution as long as it has `git`. The various live CDs we have mentioned in the past (including Kali) all provide `git` as a command or can install it to the live session.

3 Create a GIT Repository

3.1 Background

This module will demonstrate creating a Git repository from the command-line.

3.2 Process

The following steps will be carried out in a terminal. Press `ctrl+alt+t` or otherwise open a terminal through whatever means the Live Disk provides.

1. Create a directory to use as the root of your version controlled project:

   mkdir project

2. Initialize the new directory as a git repository:

   git init project

Successful output should be similar to the following:

```
Initialized empty Git repository in /home/user/project/.git/
```

3. Confirm that the new directory is a git repository:

   cd project git status

Successful output will look like the following:

```
On branch master

Initial commit

nothing to commit (create/copy files and use "git add" to track)
```

This output indicates that you are in a git repostiroy (the directory `project`) and there are no modifications in the checkout that need to be committed or saved.

If an error has occurred:

```
fatal: Not a git repository (or any of the parent directories): .git
```

If this error has occurred, make sure you changed to the target (`project`) directory before executing your `git status` command.

3.3 Conclusion

Althought it doesn't currently contain any files, you now have an empty Git repository that is capable of accepting new files and tracking changes made to those files over time.

4 Add and Modify a File

4.1 Background

As it stands, you have an empty git repository. In order to make it useful, you need to be able to add, modify, and explore files.

4.2 Process

Execute these commands from within the `project` directory.

1. Create a new file:
   echo This is a new file > file1
2. Confirm the new file exists:
   ls -l file1
3. Add the new to staging:
   git add file1
4. View the current pending changes that are still in staging:
   git status

The result should be:

```
On branch master

Initial commit

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

    new file:   file1
```

Typically, the new files will be displayed in green.

5. Commit the pending changes:
   git commit

An editor will open up where you can type in an optional commit message. Type in a message and save and quit.

For vim, press `i` then type a message, hit escape, and type `:wq`. For nano, type in your message, then press Control+O and then Control+X.

The result will look similar to the following:

```
[master (root-commit) da50d2e] Created a file.
 1 file changed, 1 insertion(+)
 create mode 100644 file1
```

This summary shows that a single file was committed that contained new (inserted) data. The file mode was set to read/write for the owner, and read for everyone else.

6. View the current status to see there are no more pending changes:
   git status

The result should be:

```
On branch master
nothing to commit, working directory clean
```

7. Confirm the commit details using the log command:
   git log

The result should be:

```
commit da50d2ea51f9bdbe244a785aa59dd87bfcd46854
Author: Stephen <stephenrd@gmail.com>
Date:   Tue Aug 23 15:59:21 2016 -0400

    Created a file.
```

4.3 Conclusion

The respository now exists with a single file in it. The file has a history that can be viwed with the `log` command, though the only history that will be visible is when the file was added to the repository. From this point on it's possible to get back to this version of the file.

5 Delete a File (and get it back)

5.1 Background

As it stands, you now have a git repository with a single file. If that file is changed, you can compare the new version with the old. And if that file is deleted, the previous version still exists.

5.2 Process

Execute these commands from within the `project` directory.

1. Delete the file, `file1`.

   rm file1

2. View the current status according to git:

   git status

The result should be:

```
On branch master
Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

    deleted:    file1

no changes added to commit (use "git add" and/or "git commit -a")
```

3. Check out the current version of the now deleted file:

   git checkout file1

4. Verify the file now exists:

   ls

The result should be:

```
file1
```

5. Confirm that there are no outstanding changes with the status command:

   git status

The result should be:

```
On branch master
nothing to commit, working directory clean
```

5.3 Conclusion

Because the file had been checked in, it was possible to check the file out again to get the version of it that exists ion the master branch of the repository.