ASSIGNMENT 3

VERSION CONTROL TOOLS

SUBMITTED BY:
RAJGAURI KHEMNAR
15IT133

SUBMITTED TO:
Ms. RAKSHA NADGIR

DATE:
23 JANUARY, 2018

WHAT IS A VERSION CONTROL TOOL:

Version control (also known as revision control or source control) is a category of processes and tools designed to keep track of multiple different versions of software, content, documents, websites and other information in development. Any system that provides change tracking and control over programming source code and documentation can be considered version control software.
The purpose of version control is ensuring that content changes under development go as planned. While version control is often carried out by a separate application, it can also be embedded into programs such as integrated development environments (IDEs), word processors, spreadsheets and, especially, collaborative web documents and pages. Version control allows servers in multiple locations to run different versions on different sites, even while those versions are being updated simultaneously.

OPEN SOURCE VERSION CONTROL TOOLS FOR SYSTEM ADMINS:

CVS:

What is CVS ?

CVS stands for Concurrent Version System. It is a version control system that has been developed in the public domain by many people beginning in 1986.
Using CVS, you can record the history of your source files (normally text files, binary files are handled with some restrictions). Instead of save every version of every file, CVS stores only the differences between the versions. CVS also helps you if you are working in a group on the same project: CVS merges the work when each developer has done its work.
Moreover, CVS allows you to isolate changes onto a separate line of development, known as a branch. When you change files on a branch, the changes do not appear on the main trunk. Later you can move the changes from one branch to another branch (or the main trunk) by merging.

Repository and Working Area

The CVS repository stores a complete copy of all the files and directories which are under version control. Normally, you never access any of the files in the repository directly. Instead, you use CVS commands to get your own copy of the files into a working directory, and then work on that copy.
CVS can access a repository by a variety of means. It might be on the local computer, or it might be on a computer across the room or across the world. Using CVS with a remote repository we talk about client/server operation. Several protocols are supported to connect to the remote repository, e.g. rsh, password authentication, GSSAPI, kerberos.
You can define several repositories if you have different development groups that work on separate projects without sharing any code. All you have to do is to specify the appropriate repository when you are starting the session.

Useful Tags

If you want to keep track of a set of revisions involving more than one file, you can use tags to give a symbolic name to a certain revision of each file.

Multiple Developers

Often, two developers try to edit the same file simultaneously. CVS supports some solutions for this situation, e.g.

- File locking or reserved checkouts: Only one person is allowed to edit each file at a time.
- Use watches: Watched files are checked out read-only. To make them read-write (and inform others watchers) the cvs edit command is used.
- Unreserved checkouts: The rarity of serious conflicts may be surprising, so often neither file locks nor watches are used. If an overlap occurs, CVS prints a warning

and the resulting file includes both versions of the lines that overlap, delimited by special markers.

A SAMPLE SESSION:

Set-up the Session

CVS commands

Set *$CVSROOT* for local access...

```
> CVSROOT=<cvs_root>
> export CVSROOT
```

... or remote access and connect to the CVS server:

```
> CVSROOT=:pserver:<username>@<servername>:<cvs_root>
> export CVSROOT
> cvs login
```

Getting the Source

CVS commands

```
> cd <work_dir>
> cvs -r checkout <module>
```

Editing the Sources

CVS commands

```
> cd <work_subdir>
> cvs admin -l <filename>
> cvs update <filename>
> cvs edit <filename>
> vi <filename>
```

Commit the Changes

CVS commands

```
> cvs commit -m "<log_message>" <filename>
> cvs unedit <filename>
```

Undo the Changes

CVS commands

```
> cvs unedit <filename>
> cvs update <filename>
> cvs admin -u <filename>
```

Cleaning Up

CVS commands

```
> cd <work_dir>
> cvs release -d <module>
```

Pros of using CVS:

- Has been in use for many years and is considered mature technology

Cons of using CVS:

- Moving or renaming files does not include a version update
- Security risks from symbolic links to files
- No atomic operation support, leading to source corruption
- Branch operations are expensive as it is not designed for long-term branching

MERCURIAL:

Mercurial is a small, powerful distributed VCS system that is easy to get started with, while still providing the advanced commands that VCS power users may need (or want) to use. Mercurial's distributed nature makes it easy to work on projects locally, tracking and managing your changes via local commits and pushing those changes to remote repositories whenever necessary.

Creating and using Mercurial repositories

Mercurial provides two basic ways of creating a local repository for a project's source code: either by explicitly creating a repository or by cloning an existing, remote repository:

- To create a local repository, use the hg init *[REPO-NAME]* command.
- To clone an existing repository, use the hg clone *REPO-NAME[LOCALNAME]* command.

Basic commands provided by Mercurial

add	-add the specified files on the next commit
annotate	-show changeset information by line for each file
clone	-make a copy of an existing repository
commit	-commit the specified files or all outstanding changes
diff	-diff repository (or selected files)
export	-dump the header and diffs for one or more changesets
forget	-forget the specified files on the next commit
init	-create a new repository in the given directory
log	-show revision history of entire repository or files
merge	-merge working directory with another revision
pull	-pull changes from the specified source
push	-push changes to the specified destination
remove	-remove the specified files on the next commit
serve	-export the repository via HTTP
status	-show changed files in the working directory
summary	-summarize working directory state
update	-update working directory

Pros:

- Easier to learn than Git
- Better documentation
- Distributed model

Cons:

- No merging of two parents
- Extension-based rather than scriptability
- Less out of the box power

<u>**GIT AND GITHUB**</u>

What is Git?

Git is a open-source code managemen tool; it was created by Linus Torvalds when he was building the Linux kernel. Because of those roots, it needed to be really fast; that it is, and easy to get the hang of as well. Git allows you to work on your code with the peace of mind that everything you do is reversible. It makes it easy to experiment with new ideas in a project and not worry about breaking anything. The Git Parable, by Tom Preston-Werner, is a great introduction to the terms and ideas behind Git.

How do I Get Set Up?

Git is pretty easy to get: on a Mac, it's probably easiest to use the git-osx-installer. If you have MacPorts installed, you may want to get Git through it; you can find instructions on the GitHub help site. (And yes, we'll talk about GitHub). On Windows, the simplest way to start rolling is to use the msysgit installer. However, if you've got Cygwin, you can git Git through there as well.

Configuration

```
git config --global user.name "Your Name"
git config --global user.email "your@email.com"
```

git init

You need to run the git init command which initializes a Git repository in that folder, adding a .git folder within it. A repository is kind of like a code history book. It will hold all the past versions of your code, as well as the current one.

git add

A commit is simply a pointer to a spot on your code history. Before we can do that, however, we need to move any files we want to be a part of this commit to the staging area. The staging area is a spot to hold files for your next commit. We can do that by using the add command:

```
git add .
```

The . simply means to add everything. You could be more specific if you wanted.

```
git add *.js
git add index.php
```

git commit

```
git commit -m "initial commit"
```

Then, -a allows you to skip the staging area. Git will automatically stage and commit all modified files when you use this option. (remember, it won't add any new files). Together, you could use these commands like this:

```
git commit -am 'update to index.php'
```

Instead of numbering them, Git uses the code contents of the commit to create a 40 character SHA1 hash. The neat part about this is that, since it's using the code to create the hash, no two hashes in your project will be the same unless the code in the commits is identical.

git status
The *git status* command allows you to see the current state of your code.

git branch / git checkout
To actually create a new branch, add the name of your new branch after the command:

```
git branch bigIdea
```

When you create a new branch, you aren't switched to it automatically. Notice that our terminal still says (master). This is where we use branches comrade command *git checkout*.

git merge
The *git merge* command is made for merging branch with master branch.

git log / gitk
Used to look at your commit history at some point during your project. This can easily be done with the log command:

```
git log
```

GitHub
Git is a great way to share code with others and work on projects together. There are a number of Git repository hosting sites, one of them being GitHub.

Open up terminal:

```
ssh-keygen -t rsa -C "your@email.com"
```

The t option assigns a type, and the C option adds a comment, traditionally your email address. You'll then be asked where to save the key; just hitting enter will do (that saves the file to the default location). Then, enter a pass-phrase, twice. Now you have a key; let's give it to GitHub.
First, get your key from the file; the terminal will have told you where the key was stored; open the file, copy the key (be careful not to add any newlines or white-space). Open your GitHub account page, scroll to SSH Public Keys, and click "Add another public key." Paste in your key and save it. You're good to go! You can test your authentication by running this:

```
ssh git@github.com
```

You'll be prompted for your pass-phrase; to avoid having to type this every time you connect to GitHub, you can automate this. I could tell you how to do this, but I'd probably inadvertently plagiarize: the GitHub Help has a plain-english article on how to do it.

git clone

To copy the entire repository to your computer.

git push

Log into GitHub and create a new repository. GitHub will give you a public clone URL (for others wanting to download your project) and a personal clone URL (for yourself).

Then, come back to your project in the terminal and give this a whirl:
git remote add origin <project url>

A remote is a project repository in a remote location. In this case, we're giving this remote a name of origin, and handing it our private clone URL.

git push origin master

This pushes the master branch to the origin remote.

git pull

When the owner pushes a new commit to the repository, you can use *git pull* to get the updates. *Git pull* is actually a combo tool: it runs *git fetch* (getting the changes) and *git merge* (merging them with your current copy).
git pull

Pros:

- Dramatic increase in operation speed
- Cheap branch operations
- Full history tree available offline
- Distributed, peer-to-peer model

Cons:

- Learning curve for those used to SVN
- Not optimal for single developers
- Limited Windows support compared to Linux

NOTE:
The version control tool Git and Github will be used by me during the implementation of my project