Name: Justin Weigle

Editing Files

1. Write the command line to use vi or vim to edit a file

   vim {file}

2. From within vim, how would you save a file

   :w

3. From within vim, how would force save a file

   :w!

4. From within vim, how you insert one file into another file

   :r {file}

5. From within vim, how you replace all commas with colons in the entire file

   :%s/,/:/g

6. How does the previous command compare with the "sed" command?

   sed "s/,/:/g" {file}

   sed is called from the command line by name.
   sed doesn't need a % before the s.
   You do need to provide sed with a file argument as well.
   Finally, if you want to update the file, you must redirect the file back into itself since sed just prints to standard out, or use the -i option.

   sed "s/,/:/g" {file} > {file}
   sed -i "s/,/:g" {file}

GIT

All of these questions, give the git command line:

1. Clone a repository:
   git clone {url}
   So for one for this class if my username were Justin
   git clone https://github.com/Justin/OperatingSystems.git
2. Create a brand new git repository:
   git init
3. Create a branch in a repo:
   git checkout -b {branchname}
4. Commit a change to a branch:
   git add -A
   git commit -m "message about commit"
5. Push a change to a remote:
   git push (then put in username and pw if applicable)
   And if the branch is not upstream
   git push --set-upstream origin {branchname}
6. Add a remote repository to a local repo:
   git remote add {remotename} {url}.git
7. Remove a remote repository from your local repo:
   git remote rm {remotename}
8. Merge changes from one repo to another:
   From within the repo you want to merge to:
   git merge {repotomerge}/master –allow-unrelated-histories
   (fix conflicts)
9. Fetch changes from a remote:
   git fetch {url} (the url is origin by default)

GIT + Steroids

All of these questions, give the git command line:

1. Write the command line (or a shell script) that will list all of the branches that have a "totest" tag (yes, Git supports tags), and for each of those branches, check out the branch, and execute a "make" to build to software, and then execute a "test" command to run the test. Just print the results.

```
#!/bin/sh

branchlist=($(git branch -a --contains tags/totest))

for i in ${branchlist[@]}
do
   git checkout $i
   make
   results=($(test))
   echo $results
done
```

2. Write the command line (or a shell script) that will list all of the branches that have a "tore-view" and create a report that shows the author, the git commits, and the changes in the files. Then, remove the tag.

```
#!/bin/sh

branchlist=($(git branch -a --contains tags/toreview))

for i in ${branchlist[@]}
do
   git checkout $i
   git log
   git diff
   git tag --delete toreview
done
```

GDB

1. How do you turn on "core files" when a program crashes?

   gdb -c {core_dump_file_name}

2. Use GDB to run a program

   gdb {program}
   (gdb) run

3. Set the command line arguments within GDB

   gdb {program}
   (gdb) run {arg1} {arg2} ... {argN}

4. Set a break point on invoking a function call

   (gdb) break {function}

5. Set a break point on a line of a source file

   (gdb) break {ln_number}

6. Set a break point whenever a value of a variable changes

   (gdb) awatch *{hex_address}
   Such as:
   (gdb) awatch *0x1266

7. Set a break point whenever a value of a variable becomes 3

   (gdb) break {location} if var == 3

8. Show the local variables of a function

   (gdb) info locals

Other Developer Tools

1. Use valgrind to find memory leaks in a C program
   Use the option --leak-check=<no|summary|yes|full> [default: summary]

2. What tool can you use to inspect the raw, binary, contents of any file?
   Hex editors/viewers such as bless, xxd, and hexdump

3. Find out how much time (real, user, and sys) that a program requires:
   /usr/bin/time -p {program} (must use full path as most shells have a built in time function that doesn't always accept options)
4. Use "strace" command to inspect all file I/O for a program such as "/bin/ls"
   strace /bin/ls

5. Use "readelf" to read the Executable and Linker Format header information for an executable (but only the header)
   readelf -h {executable}

6. Find out the run-time libraries need by a Linux ELF program and what the actual libraries that match those requirements are.
   readelf -d {executable}
   And then it is listed under NEEDED, so you can | grep "NEEDED"
   ldd is another option (ldd {executable})