Getting Started with C and UNIX

Environment Setup

CS 441/541 – Fall 2020

Project Available		Sept. 8
Component	**Points**	**Due Date (at 11:59 pm)**
Bitbucket	0	
VPN	0	
UNIX tutorial	0	
ssh to server	2	
create files	2	
git workflow	2	
process management	2	
pull request	2	
Total	10	Sept. 11
Submission	individual	Bitbucket PR

Objectives

The goal of the assignment is to make sure you have a working development environment for your programming assignments, are comfortable with using the server from the command-line, and practice the git workflow we will be using throughout the semester. Be sure to look through the many support documents and references on Canvas.

Deliverables

You will create a document with the following screenshots:

- Screenshot of ssh'ing to the class server and printing the working directory.
- Screenshot(s) of cloning the fork'd git repo, file navigation, compilation and running a C program on the class server.
- Screenshot(s) of using git to add, commit and push the changes.
- Screenshot(s) of playing with process management. Be sure to use background processes, `fg`, `ps`, `kill` and `sleep`.
- Screenshot of executing the command `top`.

The document MUST contain your name and a reasonable heading at the top, a description of each screenshot, and be in PDF format. The screenshots must be proper screenshots, not photos of a computer screen (it is up to you to figure out how to do this on your own system).

The document should be added to the repo as described below and submitted as a pull request which includes the other deliverables for the assignment.

1 Setup accounts

We will be using some tools and systems throughout the course that require login information. For each of these systems, confirm that you can login successfully, and change your password if needed. If you have trouble with any of these steps, please contact me right away to help you resolve the issue. It is very important that you establish access to these accounts early so that you can start working on the programming projects.

- BitBucket - `https://bitbucket.org/`
  BitBucket is a popular source code hosting site, similar to GitHub. If you register with the site using your `.edu` email addresses then you will have access to the unlimited version of the site. We will be using BitBucket and git in the course projects. **Deliverable: Accept my invitation to the repo(s) sent to your uwlax.edu email address.**
- UWL VPN - `https://www.uwlax.edu/its/network-telecom/vpn-virtual-private-network/`
  In order to access the class server and AutoLab from off campus, you must connect via the VPN. It is best to set this up now even if you do not anticipate needing it later. **Deliverable: Setup the VPN on your system and test it – not included in final document.**

2 Learn Linux

You will need to set up your C development environment in a UNIX system. I have provided some recommendations on how to do this that will cause the least pain when testing on the server.

The remainder of the activities you will need to do on the class server, a Linux system, from the command line. It is best to work through a tutorial first so you are familiar with the commands mentioned below.

3 Login to Class Server

You have been sent an email with account information about the class server. Use that information to login and perform the remaining tasks in this assignment. It is a good idea to read the whole document first before you do the tasks so you can plan to take the screenshots you will need.

You will need to be able to connect to the class server using ssh or sftp. The system is available to you for development and testing, and it is the environment that I will be using to grade your assignments. You will need to figure out how you will connect to the server and install any necessary software to do so. (See Canvas for suggestions.) Once you have the appropriate software, ssh into the machine and change your password.

An example of connecting to the server via ssh, changing your password and exiting. Note that the username in this example is "grace" and the server is 138.49.184.163. *Replace the server IP address, username and password with the information from the welcome email about the server.*

```
shell$ ssh grace@138.49.184.163
grace@138.49.184.163's password:
Last login: Thu Aug 6 09:00:00 2019 from 242.146.74.69
##############################################################
#           Welcome to the CS441 Course Server               #
#                                                            #
# Warning: Usage of this system may be restricted, denied, or #
# abruptly terminated during class lecture times.            #
#                                                            #
# Server status: http://138.49.184.163/                      #
##############################################################
[grace@CS441 ~]$ pwd
/home/grace
[grace@CS441 ~]$ passwd
Changing password for user grace.
Changing password for grace.
(current) UNIX password:
New password:
Retype new password:
passwd: all authentication tokens updated successfully.
[grace@CS441 ~]$
[grace@CS441 ~]$ exit
logout
Connection to 138.49.184.163 closed.
shell$
```

4 Setup a Git Repository

In this class we will be using git for distributing code for assignments, for version control while developing solutions to those assignments, and for sharing those solutions privately with me. The workflow will use forking, branching, and pull requests, but perhaps not in the same way you would on a traditional project because we want to make sure that each student (or group) has a private repo and my feedback is also private. Fortunately, git is powerful and flexible to support this workflow. It is very important that every step is done correctly otherwise it could compromise the intended privacy of the submissions and my ability to be able to evaluate your work. Remember, you must keep your work for this class in a private repo with only yourself, myself, and your partner (if you have one) with access to read or write.

Workflow in brief:

1. Fork my template repo. **WEB**
2. On your template repo, create a branch called "dev." *This will serve as the starting point of the repo so you have a branch to issue a pull request on.* **WEB**
3. Give me read and write access to your repo (my username is: ssfoley). **WEB**
4. Clone the dev branch where you like to write code. The web interface gives you a handy way to copy the command to use to do this. **WEB**
5. Work on the solution using git to help you keep track of your changes. Make sure you only push to the dev branch. Use git to also run, test, and develop on the class server. **CLI**
6. When you are done with your assignment and would like to turn it in, create a pull request from your dev branch to the master branch on your repo (thus keeping it private to just you (or your group) and me). If you want to make a change after submission, you can make a new commit and push to the dev branch and add it to the pull request. **WEB**
7. I will review the pull request and give you feedback on your work through Bitbucket and git. Your grade will be posted on Canvas.

*Note: **WEB** indicates that this part of the workflow is typically done via the web interface, whereas **CLI** indicates that this part of the workflow is typically done via the command-line interface.*

If you have been using git or when you look at references about how to use git, you will notice that we are using a combination of the forking workflow and the feature branch workflow. Even if you are familiar with git, you will want to spend some time with the Bitbucket tutorials to familiarize yourself with the user interface. `https://www.atlassian.com/git/tutorials`

4.1 Git Example

For this assignment, we will be practicing the git workflow we are using for assignments in this class using the C examples repository. We will be doing the setup and cloning on the class server, but you may also want to practice cloning it and managing changes between the server, your development environment, and bitbucket.

4.1.1 Setting up git – CLI

Before using git on a new system, you will need to configure git so that it knows who you are when you try to clone and push from/to remote repositories. After ssh'ing to the class server, execute the following commands to setup your username and email:

```
[grace@CS441 ~]$ git config --global user.name "Your Name Goes Here"
[grace@CS441 ~]$ git config --global user.email you@yourdomain.example.com
```

You only need to do this once per machine that you are using.

4.1.2 Fork the repository and create a branch – WEB

You will need to fork the examples repository I have shared with you. It will be a repository owned by me (ssfoley) and will appear in your list of repositories once I have added you to our class group. The user interface should be pretty straightforward, but they do change is fairly often, so here is a link to how to create a fork:

`https://confluence.atlassian.com/bitbucket/forking-a-repository-221449527.html`

Once you have fork'd the repo, use the web interface to create a dev branch and give me write access to the repo so I can see your pull request and merge it later. Navigate to the settings for the repo to manage the permissions for users. This is also where you would add a partner if this is a group project.

`https://confluence.atlassian.com/bitbucket/branching-a-repository-223217999.html`

4.1.3 Cloning the repo – CLI

Now that you have a repository owned by you, you can clone it on the class server. Login into the server if you aren't already, and clone the repo using the URL given to you on the web interface for bitbucket. Again, the user interface does change frequently, so below is a link. On the server you will want to use the https method.

`https://confluence.atlassian.com/bitbucket/clone-a-repository-223217891.html`

Reminder: Be sure to clone your dev branch and do all your work there.

```
[grace@CS441 ~]$ git clone https://grace@bitbucket.org/grace/cs441-examples.git -b dev
```

4.1.4 Add some files – CLI

Now you have a copy of your fork of the examples directory so you can add and edit the contents. Do the following to create a new directory and infinite loop program.

```
[ssfoley@CS441 cs441-s18-examples]$ pwd
/home/ssfoley/cs441-s18-examples
[ssfoley@CS441 cs441-s18-examples]$ mkdir proj1
[ssfoley@CS441 cs441-s18-examples]$ cd proj1
[ssfoley@CS441 testing]$ cp ../hello/hello.c ../hello/Makefile .
[ssfoley@CS441 testing]$ ls
hello.c  Makefile
[ssfoley@CS441 testing]$ mv hello.c forever.c
[ssfoley@CS441 testing]$ emacs forever.c
```

```
[ssfoley@CS441 testing]$ cat forever.c
/*
 * TODO: Fillin
 */
#include <stdio.h>

int main(int argc, char **argv) {

    printf("Hello CS441/541!!!!!\n");
    while(1){
        // infinite loop!
    }

    return 0;
}
[ssfoley@CS441 testing]$ emacs Makefile
[ssfoley@CS441 testing]$ cat Makefile

CC=gcc
CFLAGS=-Wall -O0 -g

all: forever

forever: forever.c
        $(CC) forever.c $(CFLAGS) -o forever

clean:
        rm forever *.o
[ssfoley@CS441 testing]$ make
gcc forever.c -Wall -O0 -g -o forever
```

Now that we have made some changes to the repository we need to make sure git is aware of them and will help us save and coordinate these changes across all clones of the repository. It is always a good idea to use `git status` to check the status of your files.

```
[ssfoley@CS441 testing]$ git status
# On branch dev
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#       ./
nothing added to commit but untracked files present (use "git add" to track)
[ssfoley@CS441 testing]$ git add .
[ssfoley@CS441 testing]$ git status
# On branch dev
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
```

```
#       new file:   Makefile
#       new file:   forever
#       new file:   forever.c
#
[ssfoley@CS441 testing]$ git commit -m "adding a new directory and some testing code"
[dev 313bde5] adding a new directory and some testing code
 3 files changed, 25 insertions(+)
 create mode 100644 proj1/Makefile
 create mode 100755 proj1/forever
 create mode 100644 proj1/forever.c
[ssfoley@CS441 testing]$ git status
# On branch dev
# Your branch is ahead of 'origin/dev' by 1 commit.
#   (use "git push" to publish your local commits)
#
nothing to commit, working directory clean
[ssfoley@CS441 testing]$ git push
...
Password for 'https://ssfoley@bitbucket.org':
Counting objects: 7, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (6/6), done.
Writing objects: 100% (6/6), 3.37 KiB | 0 bytes/s, done.
Total 6 (delta 1), reused 0 (delta 0)
To https://ssfoley@bitbucket.org/ssfoley/cs441-s18-examples.git
   7f9ddb2..313bde5  dev -> dev
[ssfoley@CS441 testing]$ git status
# On branch dev
nothing to commit, working directory clean
```

Deliverable: screenshot(s) of (1) creating the directory, copying files, and the contents of forever.c and Makefile, (2) git status, git add, git commit, git status, git push workflow. Be sure that the image(s) are readable.

(Optional) update the .gitignore file at the top-level of the repo to ignore the new executable we created, and the other executables from the other directories. .gitignore is another file whose changes must be added, committed and pushed.

5 Managing Processes

Process management via the command line is important to know for general navigation as well as to understand how the topic of process management in class is implemented in Linux. We will be using the infinite loop program you wrote in the previous step to explore how to run processes, view running processes, and kill them.

Processes can be run in the foreground or in the background. Typically, you execute a command and wait for it to finish, with the output being displayed as it arrives. You can run multiple programs on the same line in the foreground (one after the other in sequence) using the ';'. You can run multiple programs on the same line in the background (concurrently) by using the &. Play with this to get a sense for how and when the processes run. You may want to have another ssh connection to the server; one for running experiments and the other for observing the results. Below are some experiments to try. If you don't know how the commands work, use `man` to view the manual pages. **Be sure to clean up any running processes on the server after doing these experiments!**

Note: ^C is how we denote typing Control+C.

```
[ssfoley@CS441 testing]$ ./forever
Hello CS441/541!!!!!
^C
[ssfoley@CS441 testing]$ ./forever &
[1] 4380
[ssfoley@CS441 testing]$ Hello CS441/541!!!!!

[ssfoley@CS441 testing]$ ps
  PID TTY          TIME CMD
 4277 pts/0    00:00:00 bash
 4380 pts/0    00:00:05 forever
 4381 pts/0    00:00:00 ps
[ssfoley@CS441 testing]$ ./forever & ./forever & ./forever &
[2] 4382
[3] 4383
[4] 4384
Hello CS441/541!!!!!
[ssfoley@CS441 testing]$ Hello CS441/541!!!!!
Hello CS441/541!!!!!
ps
  PID TTY          TIME CMD
 4277 pts/0    00:00:00 bash
 4380 pts/0    00:00:25 forever
 4382 pts/0    00:00:03 forever
 4383 pts/0    00:00:03 forever
 4384 pts/0    00:00:03 forever
 4385 pts/0    00:00:00 ps
[ssfoley@CS441 testing]$ kill 4382
[ssfoley@CS441 testing]$ ps
  PID TTY          TIME CMD
 4277 pts/0    00:00:00 bash
 4380 pts/0    00:00:43 forever
```

```
 4383 pts/0    00:00:21 forever
 4384 pts/0    00:00:21 forever
 4386 pts/0    00:00:00 ps
[2]   Terminated              ./forever
[ssfoley@CS441 testing]$
```

Sleep is a handy command to run that runs for a while, but also terminates. Here is an example of a small experiment.

```
[ssfoley@CS441 testing]$ sleep 10
[ssfoley@CS441 testing]$ sleep 50 & sleep 50 & sleep 20 &
[5] 4388
[6] 4389
[7] 4390
[ssfoley@CS441 testing]$ ps
  PID TTY          TIME CMD
 4277 pts/0    00:00:00 bash
 4380 pts/0    00:01:48 forever
 4383 pts/0    00:01:26 forever
 4384 pts/0    00:01:26 forever
 4388 pts/0    00:00:00 sleep
 4389 pts/0    00:00:00 sleep
 4390 pts/0    00:00:00 sleep
 4391 pts/0    00:00:00 ps
[ssfoley@CS441 testing]$ fg
sleep 20
[ssfoley@CS441 testing]$ ps
  PID TTY          TIME CMD
 4277 pts/0    00:00:00 bash
 4380 pts/0    00:02:06 forever
 4383 pts/0    00:01:44 forever
 4384 pts/0    00:01:44 forever
 4388 pts/0    00:00:00 sleep
 4389 pts/0    00:00:00 sleep
 4392 pts/0    00:00:00 ps
[ssfoley@CS441 testing]$ ps
  PID TTY          TIME CMD
 4277 pts/0    00:00:00 bash
 4380 pts/0    00:02:37 forever
 4383 pts/0    00:02:15 forever
 4384 pts/0    00:02:15 forever
 4395 pts/0    00:00:00 ps
[5]-  Done                    sleep 50
[6]+  Done                    sleep 50
```

5.1 top

Another command to run is top. Also include a screenshot of running this command.

5.2 Pull Request

Finally, you will need to add your PDF of your assignment to the repository and submit a pull request.

5.2.1 Creating a Pull Request

Make sure all of your commits and changes are visible on the web interface for Bitbucket. You will then make a pull request from your dev branch to your master branch. Remember that your master branch should be the one that is the same as the template I shared with you when you fork'd the repo.

`https://www.atlassian.com/git/tutorials/making-a-pull-request`