

CprE 288 – Introduction to Embedded Systems (Programming Style, Doxygen, SVN)

Instructors:
Dr. Zhao Zhang (Sections A, B, C, D, E)
Dr. Phillip Jones (Sections F, G, J)

1

Overview of This Week

- Announcements
- Lab 9
- Homework Review
- Suggested Programming Style for Lab Project
- Doxygen: Automatic Documentation Generation
- SVN: Version Control System

2

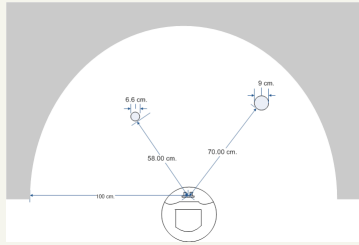
Announcement

- Homework 8 due on Thursday in the class
- **Exam 2 next Thursday**
 - Open book, open notes
 - Coverage: USART, ADC, Input Capture, Output Compare

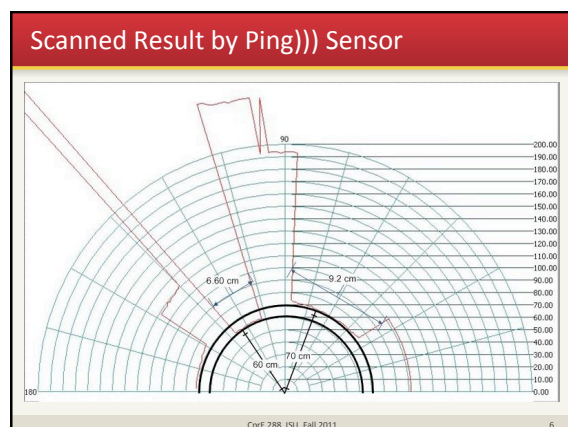
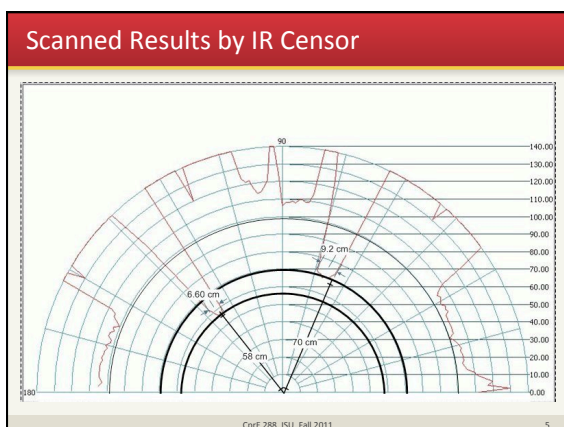
3

Lab 9: Object Counting and Size Discrimination

How do you distinguish two objects of different size?



4



Data Analysis

How can your program identify and distinguish different objects from the following raw data?

Degrees	IR Distance (cm)	Sonar Distance (cm)
0	120	324
2	123	330
4	119	363
6	40	40
8	40	40
10	40	41
... (more)		

CprE 288, ISU, Fall 2011

7

Data Analysis

Step 1: Scan the array to identify gaps, convert them to **angular sizes**

- What's your algorithm?

Step 2: For each object, convert its **distance** and **angular size** into **linear size** (width)

- What's your mathematic formula?

CprE 288, ISU, Fall 2011

8

Homework Review

- HW5, USART
- HW6, ADC
- HW7 and HW8 will be reviewed next week

9

Suggested Programming Style for Lab Project

References and Readings

- [GNU Coding Standards](#). Free Software Foundation
- [Proper Linux Kernel Coding Style](#). Greg Kroah-Hartman, Linux Journal, July 01, 2002
- [Recommended C Style and Coding Standards](#). L. W. Cannon et al.
- [Indent Style](#), Wikipedia

Credit: Jafar M. Al-Kofahi made contribution to an early version of 288 Lab Project Coding Style

CprE 288, ISU, Fall 2011

10

Suggested Programming Style for Lab Project

You are suggested to use the Programming style presented in this lecture

- It's a simplified version of [GNU Coding Standards](#), with elements from the other references
- You may choose some variants, if with good reason

ALL partners of the same project team must use the same style with the same variants

CprE 288, ISU, Fall 2011

11

Why do we need it?

```

int
i;main(){for(;i["]<i;++i){--i;"};read('-'-',i+++"hell\
world!\n",'/'/'/');}read(j,i,p){write(j/p+p,i---j,i/i);}
-- Dishonorable mention, Obfuscated C Code Con-
test, 1984.
Author requested anonymity.
```

From "Recommended C Style and Coding Standards"

CprE 288, ISU, Fall 2011

12

Why do we need it?

```
int m1 (char *p, int width)
{
    int r = 0;
    char c;

    while (width--)
    {
        c = *p++;
        if (c == 0)
            break;
        if (c == ' ')
            continue;
        if (c < '0' || c > '7')
            return -1;
        r = r * 8 + (c - '0');
    }
    return r;
}
```

Credit: Jafar M. Al-Kofahi

CprE 288, ISU, Fall 2011

13

Why do we need it?

```
int getOctal (char *chrValue, int intWidth)
{
    int intResult = 0;
    char chrTmp;

    while (intWidth--)
    {
        chrTmp = *chrValue++;
        if (chrTmp == 0)
            break;
        if (chrTmp == ' ')
            continue;
        if (chrTmp < '0' || chrTmp > '7')
            return -1;
        intResult = intResult * 8 + (chrTmp - '0');
    }
    return intResult;
}
```

Credit: Jafar M. Al-Kofahi

CprE 288, ISU, Fall 2011

14

Why do we need it?

We need a good coding style for many reasons

- Understand the code written by ourselves after some time
- Let others understand the code
- Reduce the number of bugs and the [debugging time](#)
- Overall, reduce the time spent on 288 Lab Project

CprE 288, ISU, Fall 2011

15

C Programming Style

From GNU Coding Standards, Ch. 5, "Making the Best Use of C"

- Formatting: Format your source code
- Comments: Commenting your work
- Syntactic Convention: Clean use of C Constructs
- Names: Naming variables, functions, and files

CprE 288, ISU, Fall 2011

16

Program File Layout

Suggested layout for .c files

1. A prologue that tells what is in the file
2. Any header file includes
3. Any defines and typedefs
4. Global data declarations
5. Functions, in some meaningful order

More details: Recommended C Style and Coding Standards, Sec 2.2 Program files

CprE 288, ISU, Fall 2011

17

Program File Layout: Example

```
/*
 * ping.c: Ping() sensor related functions
 */
Prologue

#include <avr/io.h>
#include <avr/interrupt.h>
#include "servo.h"
Includes

// Number of clock cycles for 1-meter distance (single-trip) under prescaler 256
#define TICKS_PER_METER 735
Defines and defs

volatile unsigned falling_time; // captured time of falling edge
volatile unsigned rising_time; // captured time of rising edge
Global variables

unsigned ping_read()
{
    ...
}
Functions
```

CprE 288, ISU, Fall 2011

18

Header File Layouts

Use the same layout for .c program files, for **declarations visible to outside**

Use C Macro def to avoid nested includes

```
#ifndef EXAMPLE_H
#define EXAMPLE_H
... /* body of example.h file */
#endif /* EXAMPLE_H */
```

Use extern for global variable visible to outside

```
extern int sound_speed;
```

CprE 288, ISU, Fall 2011

19

Format Function

GNU Function layout

- Brace starts at column 1 of a new line
- Function name starts at column 1 of a new line

```
static char *
concat (char *s1, char *s2)
{
    ...
}
```

CprE 288, ISU, Fall 2011

20

Format Expression

Break an long expression: Split it **before** an operator and align the two parts properly

```
if (foo_this_is_long && bar > win (x, y, z)
    && remaining_condition)
```

Extra parenthesis: Add extra parentheses if they can make expressions clearer

```
max = (x > y) ? x : y;
```

CprE 288, ISU, Fall 2011

21

Indent Style: GNU

```
int
sample_func()
{
    while (x == y)
    {
        something ();

        if (some_error)
            do_correct ();
        else
            cont_as_usual ();
    }

    finalthing ();
}
```

GNU indent style

- The opening brace occupies a line
- The opening brace is indented by 2 spaces
- The next statement is indented by another 2 spaces

CprE 288, ISU, Fall 2011

22

Indent Style: K&R

```
int sample_func()
{
    while (x == y) {
        something();

        if (some_error)
            do_correct();
        else
            cont_as_usual();
    }

    finalthing();
}
```

K&R indent style

- The opening brace of a control body does NOT take a line
- The next statement is indented by 4 spaces

The K&R Book: The C Programming Language, Brian W. Kernighan and Dennis M. Ritchie

CprE 288, ISU, Fall 2011

23

Indent Style: Allman

```
int sample_func()
{
    while (x == y)
    {
        something();

        if (some_error)
            do_correct();
        else
            cont_as_usual();
    }

    finalthing();
}
```

Allman indent style (ANSI style)

- The opening brace of a control body takes a line
- The opening brace is indented by 0 space
- The next statement is indented by 4 spaces

CprE 288, ISU, Fall 2011

24

Indent Style: Simple Control Statements

GNU:

```
if (x == y)
    do_something ();
else
    do_others ();
```

K&R and Allman:

```
if (x == y)
    do_something();
else
    do_others();
```

If the control body is a single statement:

- GNU: Indented by 2 spaces
- K&R and Allman: Indented by 4 spaces

GNU function call: Note the extra space between the function name and "("

CprE 288, ISU, Fall 2011

25

Indent Style: Lab Project

Which style to use? Your choice!

- Each style has its own rational and history

For the Lab Project

- GNU is more generous in using line space, more popular today because of GNU projects
- Allman is the most compatible, among the three, with the AVR's studio's default indentation
- K&R is the most compact, and more AVR-compatible than GNU

Everyone in the same team must use the same style!

CprE 288, ISU, Fall 2011

26

Format Switch Statement

```
switch (expr)
{
    case ABC:
    case DEF:
        statement;
        break;
    case UVW:
        statement;
    case XYZ:
        statement;
        break;
}
```

GNU Style:

- Cases are aligned with the opening brace (indented by 2 spaces)
- The statements are indented by 2 spaces from case, 4 spaces from switch

CprE 288, ISU, Fall 2011

27

Format Switch Statement

```
switch (expr) {
    case ABC:
    case DEF:
        statement;
        break;
    case UVW:
        statement;
    case XYZ:
        statement;
        break;
}
```

K&R Style

- Cases are aligned with the switch
- Statements are indented by 4 spaces from case and switch

CprE 288, ISU, Fall 2011

28

Format Switch Statement

```
switch (expr)
{
    case ABC:
    case DEF:
        statement;
        break;
    case UVW:
        statement;
    case XYZ:
        statement;
        break;
}
```

Allman Style*

- Cases are aligned with the switch and the open brace
- Statements are indented by 4 spaces from case and switch

* This may not be the original Allman style

CprE 288, ISU, Fall 2011

29

Format Statement

Automatic indent tool: indent

- Available on Linux, Mac or other UNIX-type systems

Format with the GNU style

indent -gnu sample.c

Format with the K&R style

indent -kr sample.c

Format with the original Berkeley style (also popular)

indent -orig sample.c

CprE 288, ISU, Fall 2011

30

Commenting Your Work

GNU guidelines and our suggestion:

- Each **program** should start with a comment saying briefly what it is for
- Each **function** should have a starting comment saying what the function does
- Explain **arguments** properly, particularly if there is anything unusual
 - E.g. A string that is not necessarily zero-terminated
- Explain the **return value**
- **Be generous** in commenting, try to put a comment for every block of statements or statement with non-straightforward meaning

CprE 288, ISU, Fall 2011

31

Commenting Your Work

More from “Recommended C Style and Coding Standards”

- Write a block of **comment prologue** to **each function**
- Make function **return value have its own line**, with probably a comment explain the return value (same as GNU)
- Try to **align comments**
- Use a blank line between local variable declarations and the function’s statements

CprE 288, ISU, Fall 2011

32

Commenting: Example

```
/* Move servo to a angular position given by degree. */
void
move_servo(unsigned degree)
{
    unsigned pulse_width;          // pulse width in Timer/Counter cycles

    // Pulse width is (1+(degree/180))*t cycles, t is number of clock cycles per millisecond
    pulse_width = 1*MS_TCKS + (degree*MS_TCKS/180);

    OCR3B = pulse_width-1;         // set pulse width
    wait_ms(500);                  // wait for half second for servo to settle
}
```

CprE 288, ISU, Fall 2011

33

Commenting: Example

```
/* Start Ping))) sensor, read the pulse width, and return distance in millimeter */
unsigned          // return distance, 0 if out of range (> 1000mm)
ping_read()
{
    send_pulse();    // send the starting pulse to PING
    state = LOW;     // now in the LOW state

    // Enable Timer1 and interrupt, with noise cancellation (ICNC=1),
    // detecting rising edge (ICES=1), and prescaler 1024 (CS=101)
    TCCR1B = _BV(ICNC) | _BV(ICES) | _BV(CS2) | _BV(CS0);
}
```

CprE 288, ISU, Fall 2011

34

Commenting: Example

```
// Wait until IC is done
while (state != DONE)
{
}

// Disable Timer/Counter 1: CS=000
TCCR1B &= ~(_BV(CS2) | _BV(CS1) | _BV(CS0));

// Convert time difference in cycles to distance in millimeter
unsigned dist = (falling_time - rising_time) / (2 * cycles_per_mm);

// Out of range?
if (dist > 1000)
    dist = 0;

return dist;
}
```

CprE 288, ISU, Fall 2011

35

Commenting: Doxygen

Note: You are required to use **Doxygen**, which has all those requirements included, plus special formats for commenting

Doxygen is a **documentation generator**

- It generates software reference documents automatically from your comments

CprE 288, ISU, Fall 2011

36

Nested Control Statement

Always use braces to separate nested control statements

The following style is bad

```
if (foo)
{
    if (bar)
        win ();
    else
        lose ();
}
```

CprE 288, ISU, Fall 2011

37

Naming Conventions

GNU coding standards:

Use underscore to separate multiple words

```
falling_time
rising_tme
init_servo
move_servo
```

Try to use short local variable names

CprE 288, ISU, Fall 2011

38

Naming Conventions

More from "Recommended C Style and Coding Standards"

- Avoid local declarations that override declarations at higher level, e.g. local vs. global, same local names in nested blocks
- Avoid using names started with underscore (to avoid conflicts with system/library variables)
- #define constants should be in all CAPS
- Function, typedef, and variable names, as well as struct, union, and enum tag names should be in lower case
- Avoid names close to each other, e.g. foo and Foo, foobar and foo_bar, bl and b1 and bl (with upper case l)

CprE 288, ISU, Fall 2011

39

White Space

Use white spaces generously

```
if ((a + b) == (c - d))
```

Split long for-loop and align the lines

```
for (curr = *listp, trail = listp;
    curr != NULL;
    trail = &(curr->next), curr = curr->next)
{
    ...
}
```

CprE 288, ISU, Fall 2011

40

Program File Organization

Use multiple program files, one .c file and one .h file for each program module

Examples:

```
lcd.c, lcd.h
util.c, util.h
ir_sensor.c, ir_sensor.h
ping.c, ping.h
robot.c, robot.h
servo.c, servo.h
main.c
```

CprE 288, ISU, Fall 2011

41

Doxygen Introduction

Dr. Zhao Zhang and Dr. Phillip Jones
Based on Doxygen Tutorial by Jafar Al-Kofahi

CprE 288, ISU, Fall 2011

42

Why Use Doxygen

To generate software reference documentation **automatically**

- So you don't have to write a separate document for your project!

It works with C, C++, Java, Fortran, C# and many other languages

Documentation is extracted from special comments in the source code

CprE 288, ISU, Fall 2011

43

File Headers

Add a file header as follows

```
/**
 *   @file util.h
 *   @brief this header file will contain all required
 *          definitions and basic utilities functions.
 *
 *   @author: Dr. Zhao Zhang
 *
 *   @date 2/28/2009
 */
```

CprE 288, ISU, Fall 2011

44

File Headers

Doxygen commands:

- @file:** declares the file name
- @brief:** gives a brief description of the file
- @author:** gives the author name
- @date:** specifies the date of creation

CprE 288, ISU, Fall 2011

45

Function Example

```
/**
 * Print a single character to the LCD.
 * @author      Dr.Zhao Zhang
 * @param ch    The character to print
 * @date        2/28/2009
 */
void
lcd_putc(char ch)
{
    // Sends out the higher half to LCD
    PORTA |= ch >> 4;
    lcd_toggle_clear(1);

    // Sends the lower half to LCD
    PORTA |= ch & 0x0F;
    lcd_toggle_clear(1);
}
```

CprE 288, ISU, Fall 2011

46

Function Example

```
/**
 * This method will be used to print a string to the lcd.
 * @author      Jafar Al-Kofahi
 * @param str    The string to print on the LCD
 * @date        2/28/2009
 */
void
lcd_puts(char *str)
{
    // Write characters in str to LCD one by one until NULL is met
    while (*str)
        lcd_putc(*str++);
}
```

CprE 288, ISU, Fall 2011

47

Comment Functions

Add a comment prologue to each function

- @author:** Gives the author name
- @date:** Specifies the date of creation
- @param:** Describe a parameter
- @return:** Describe a return value

CprE 288, ISU, Fall 2011

48

Tutorial and Example

A tutorial:

<http://class.ee.iastate.edu/cpre288/lecture/Doxygen/DoxygenTutorial.pdf>

Example Doxygen output:

<http://class.ee.iastate.edu/cpre288/lecture/Doxygen/html/files.html>

CprE 288, ISU, Fall 2011

49

SVN Introduction and Operations

Dr. Zhao Zhang and Dr. Phillip Jones

CprE 288, ISU, Fall 2011

50

SVN Introduction: Outline

- Introduction
- Provided functionality
- SVN Work-flow
- Common problem
- Subversion (SVN)
- SVN Limitation
- Current available systems
- Tutorial

Credit: Jafar M. Al-Kofahi prepared an early version of this lecture

51

Introduction

- What is it?
- SVN is a [version control system](#) that can be used for [concurrent development](#) of program files, and it [manages different versions](#) of those files in a repository.
- It is commonly used by software companies to keep track of different versions of their projects.

52

Introduction

What is it being used for?

- Maintaining file history.
- Managing collaboration on files.
- Managing releases.

53

Common Functionality

- Check-out
- Check-in (commit)
- Update
- Add/Remove
- History
- Compare

54

SVN – Workflow

- Get a working copy of the project/file
- Make changes in your copy
- **Test** them locally
- Integrate them with any changes made to the [Repository](#)
- Commit them back to the Repository
- Repeat these steps until a release is ready
- Start making changes again for next release

55

SVN – Commit

When you commit your changes you should do the following:

Try always to group your committed changes, for example if you have 5 new files, and 3 of them were added to support functionality A, then those 3 should be committed together and you **MUST** write one log for them.

For the remaining two if they were both standalone (their changes does not affect any other files) each one of them **MUST** get a separate log.

56

SVN – Commit

What to write in the log?

- Any major change you did
- Why you did it
- Any reported bugs and fixes
- Any new methods, files, sub-classes..etc
- ...etc

57

Common problem

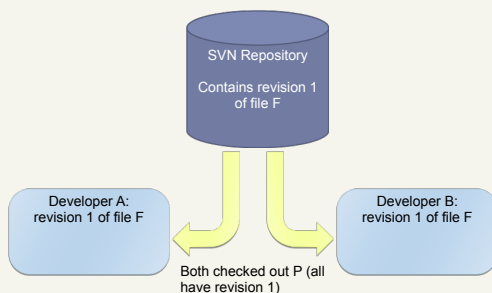
There are couple of problem scenarios, and in the next couple of slides we will talk about one of them.

Consider the following scenario:

- We have one project called P that contain file F.
- Two new developers A and B joined the team.
- Both were assigned the task of changing file F.

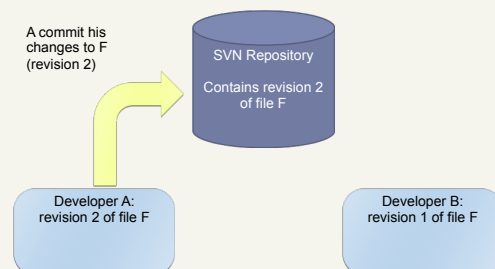
58

Problem - 2



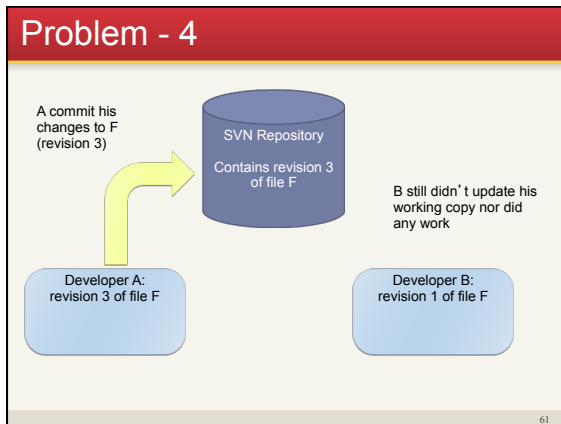
59

Problem - 3

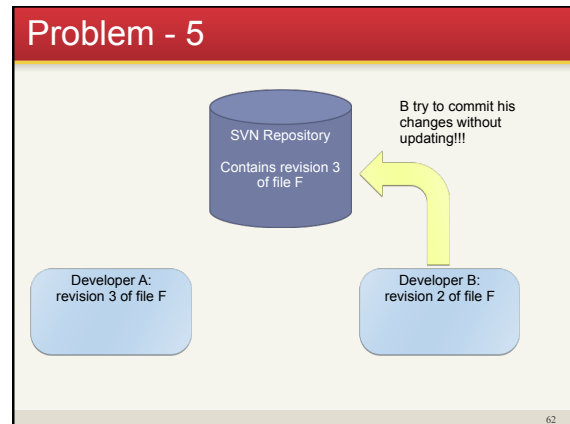


60

Problem - 4



Problem - 5



Problem - 6

Because of B commit there will be a **conflict** problem, and there are two cases:

B's work is totally independent of A's work (separate), which is easily solved by merging them.

B and A both modified the same code portion!
What do you think should happen?

A preventive solution is Exclusive locks.

Concurrent Versioning Systems (CVS)

- CVS is very similar to SVN, and usually the differences between them is in term of architecture (different kind of repository, information stored ...etc).
- SVN systems started around 2000, while CVS systems existed around 1985.
- Some argue that SVN is the new successor of CVS, that solve CVS problems, while others consider it as a separate system.

Current available systems

Some websites provides a CVS/SVN server:
www.sourceforge.net

There is also one for ECE department that you will be using for managing your code
www.source.ece.iastate.edu

Some come as clients that can be used locally or can be connected to a server:
TorToiseSVN
Subclipse (a plug-in for Eclipse)

References

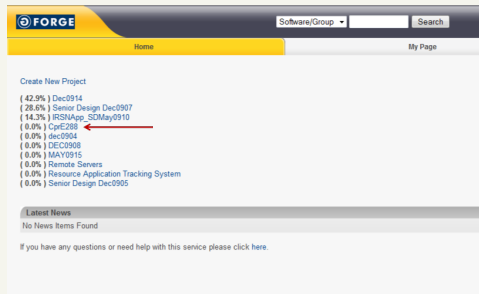
1. Ben Collins-Sussman, Brian W. Fitzpatrick, and C. Michael Pilato, Version Control with Subversion: For Subversion 1.5: (Compiled from r3305), 2008
2. Student Technology Services, Johns Hopkins University.
<http://sts.jhu.edu/loader.php?page=tutorial-subversion.html>
3. Jennifer Vesperman,
http://www.linuxdevcenter.com/pub/a/linux/2002/01/03/CVS_intro.html
4. www.Wikipedia.org
5. http://www.pushok.com/soft_svn_vscvs.php

SVN Operations: MyGForge and Clients Tutorial

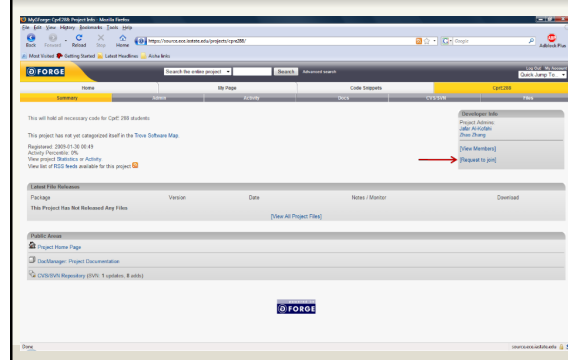
Create an account

- Before we start, you need to go to <https://source.ece.iastate.edu/>
- Then do a log-in using your Net-ID
- Then, under the Home tab select your project (for example "CprE288") and submit a "Request to join", type in your name and Net-ID and press the submit button.

Create an account



Create an account



Create an account

You can request to join a project by clicking the submit button. An administrator will be emailed to approve or deny your request.

If you want, you can send a comment to the administrator:

Name : Student Name
Net-ID : net-123

Submit

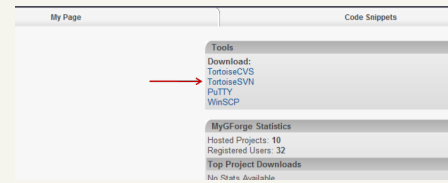
Using the website

- The website will provide you with the SVN basic functionalities, such as:
 - Browse SVN files: see the contents in the SVN for any revision
 - Compare revisions: see differences between revisions
 - Recent activity: get informed by any recent changes to the project contents, reported bugs...etc

Using the TorToiseSVN

- The TorToiseSVN tool does not have an interface, it will be operated by using Windows explorer. The tool will give you the ability to Check-out the project code, commit your changes, update your working copy, and more.
- A link to the tool installer can be found at the website, under Home tab, to the right side of the screen under Tools.
- You **MUST** be registered with the project before using the client tool.

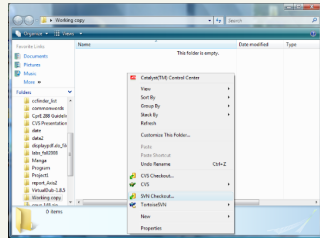
Using the TorToiseSVN



Make sure you select TorToiseSVN and NOT TorToiseCVS, the CVS tool will not work with our projects since it is CVS (Concurrent Versioning System) and not SVN (Subversion)

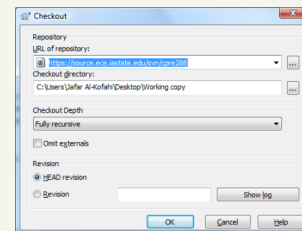
Using the TorToiseSVN

- After installing the tool, create a folder on your local drive and name it whatever you like, then open it and do a right-mouse click within it. And select SVN Checkout.



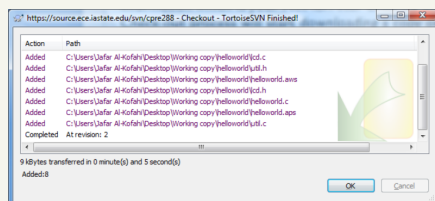
Using the TorToiseSVN

- Then put the given project URL for the repository: (e.g. <https://source.ece.iastate.edu/svn/cpre288>) And click OK.



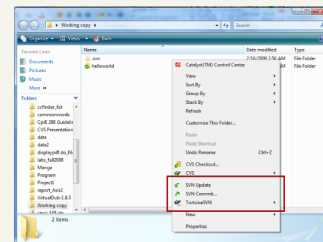
Using the TorToiseSVN

- After pressing Ok, it will ask you for your account information which is your Net-ID. After entering them the Check-out process will start downloading a copy of the project to your local folder.

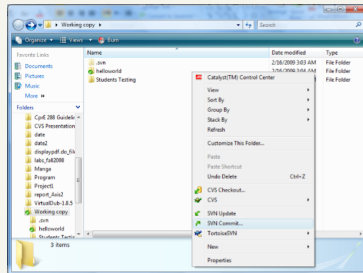


Using the TorToiseSVN

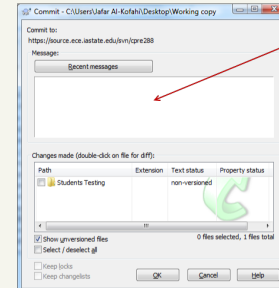
- Now after you got your copy, you can start using the SVN functionality (commit to apply changes, add files and folders, update local copy...etc)



Using the TortoiseSVN – Adding new folder



Using the TortoiseSVN - Commit



Write a log to describe your committed changes

Using Unix Shell or MS DOS

- Both use the same command and have the same options.
- To do any client operation on your SVN use the `svn` command.
- To view all available commands you can use the command: `svn help`

Using Unix Shell or MS DOS

- For this course we are only interested in the following commands (to know more about each command execute `svn command -?`):
 - Check-out:
 - First go to the folder where you want your project to be checked out to (for the first time).
 - execute `svn checkout --username developername https://source.ece.iastate.edu/svn/projectname`
 - Where *developername* is your Net-ID and *projectname* is your assigned project name.

Using Unix Shell or MS DOS

- e.g. `svn checkout --username jafar https://source.ece.iastate.edu/svn/cpre288`
- After issuing this command you will be prompted with security confirmation message, please accept it by pressing `t` for temporarily accepting it or `p` for permanently accepting it. Notice that temporarily accepting it will keep prompting this message every time you do an operation on the SVN

Using Unix Shell or MS DOS

Screen shot of executing a check-out command on DOS

```
C:\Documents and Settings\Jafar>svn checkout --username jafar https://source.ece.iastate.edu/svn/cpre288
Error validating server certificate for 'https://source.ece.iastate.edu:443':
 - The certificate is not issued by a trusted authority. Use the
   fingerprint to validate the certificate manually!
Certificate information:
- Issuer: source.ece.iastate.edu
- Valid: From Fri, 31 Oct 2008 01:08:49 GMT until Sat, 31 Oct 2009 01:08:49 GMT
- Subject: cpre288
- Fingerprint: 481b63533919fc2f461a82f37701a119708e8755e75e9
[Reject] - accept (temporarily) or accept (permanently)? t
Authentication realm: <https://source.ece.iastate.edu:443> Document repository
Password for 'jafar': *****
A cpre288\helloWorld
A cpre288\helloWorld\led.c
A cpre288\helloWorld\util.h
A cpre288\helloWorld\helloWorld.asm
A cpre288\helloWorld\led.h
A cpre288\helloWorld\helloWorld.c
A cpre288\helloWorld\helloWorld.apa
A cpre288\helloWorld\util.c
Checked out revision 2.
```

Using Unix Shell or MS DOS

- Commit (check-in):
 - Go to the project folder
 - To commit all of your changes, execute `svn commit --message "log message"` or you can use the file option if you have your log written to a text file `svn commit --file fileName`
- Add/Delete
 - You can use the command to add/delete file or folders, but you need to give the path to the file/folder you want to add/delete from your current path.
 - To execute an add command `svn add path`
 - Same thing for the delete command `svn delete path`

Using Unix Shell or MS DOS

- Update:
 - To update your local copy from the SVN
 - What will be updated is all files that do not have a conflict with the SVN (missing files will be restored, but changed files will not be restored)
 - `svn update`
- Revert
 - To revert a changed file in your local copy back to the current SVN revision. Or to restore a deleted file from your SVN version (e.g. after executing the delete command)
 - `svn revert filepath`

Using Unix Shell or MS DOS

- Status:
 - To check the difference between your local copy and the SVN. The first character will be one of the following:

Character	Meaning
?	File is not under version control
!	File is missing
A	File is scheduled for addition
C	File has textual conflicts from an update
D	File is scheduled for deletion
M	The contents of the file has local modifications

- `svn status`

Using Unix Shell or MS DOS

- Differences:
 - To show differences between files (e.g. local copy and SVN)
 - `svn diff` : will show the differences between all files/folders from the path the command where executed from.
 - To do it for a specific folder/file `svn diff path`
 - To output the results into a file instead of the console `svn diff > outputFile`

Reference

- Go to: <http://svnbook.red-bean.com/en/1.5/svn-book.pdf>
- For more information about SVN
- You MUST read **chapter 2** for more knowledge about SVN basic commands