ECE 2400 Linux, Git, C/C++ Cheat Sheet

Linux Commands

`# comment`	comment, does nothing
`man command`	display help for given `command`
`echo "string"`	display given `string`
`echo "string" > file`	create `file`
`cat a`	display file `a`
`less a`	display file `a` with paging and search
`ls`	list contents of current working dir
`ls -la`	list contents of current working dir (verbose)
`ls A`	list contents of dir `A`
`ls *.txt`	list files with `.txt` suffix in current working dir
`pwd`	display current working dir
`mkdir A`	make dir `A` to `B`
`mkdir -p A/B`	make all dirs in path `A/B`
`cd A`	change current working dir to `A`
`cd ..`	change current working dir to parent dir
`cd ~`	change current working dir to home dir
`tree`	recursively list contents of current working dir
`cp a b`	copy file `a` to `b`
`cp -r A B`	copy dir `A` to `B`
`mv a b`	move file `a` to `b`
`mv A B`	move dir `A` to `B`
`rm a`	remove file `a`
`rm -r A`	remove dir `A`
`wget url`	download file at `url`
`grep "string" a`	search file `a` for given string
`grep -r "string" A`	recursively search files in dir `A`
`find . -name "string"`	find files named `string` in dir `.`
`tar -czvf a.tgz A`	create archive `a.tgz` of dir `A`
`tar -xzvf a.tgz`	extract archive `a.tgz`
`top`	view what is running on system
`ENVVAR="string"`	set environment variable
`echo ${ENVVAR}`	display given environment variable
`cmd > a`	redirect output of `cmd` to newly created file `a`
`cmd >> a`	redirect output of `cmd` to append to file `a`
`cmd_a && cmd_b`	execute `cmd_a` and then execute `cmd_b`
`cmd_a | cmd_b`	send output from `cmd_a` to `cmd_b`
`source setup-ece2400.sh`	source setup script for course
`quota`	check disk usage
`trash`	move file to `${HOME}/tmp/trash`

git Commands

`help cmd`	display help on git command `cmd`
`clone url`	clone repo at given URL
`add a`	add file `a` to index
`add A`	add directory `A` to index
`add -u`	add all tracked files to index
`commit`	commit indexed files
`commit -a`	commit all tracked files
`commit -m "msg"`	commit indexed files w/ commit `msg`
`log`	show history log of previous commits
`status`	show status of local repo
`checkout a`	revert file `a` to last commit
`checkout A`	revert dir `A` to last commit
`pull`	pull remote commits to local repository
`push`	push local commits to remote repository
`whatchanged`	show incremental changes for each commit
`xstatus`	compact status display
`xlog`	compact log display
`xadd`	add all tracked, modified files to index
`xpull`	short for `pull --rebase`

gcc/g++ Command Line Options

`-o bin`	output binary file name
`-c`	compile intermediate object file
`-Wall`	turn on all warnings
`-O3`	turn on optimizations

gdb Commands

`break loc`	set a breakpoint at location `loc`
`run`	start running program
`record`	start recording for reverse debugging
`step`	execute next C statement, step into function
`next`	execute next C statement, do not step into function
`rs`	reverse step, undo current C statement
`print var`	print C variable `var`
`continue`	continue on to next breakpoint
`refresh`	refresh source code display
`quit`	exit GDB

Use the first few letters of the command as a short-cut as long as these letters uniquely distinguish the command. For example, you can use `b` for `break`, `s` for `step`, `n` for `next`, and `c` for `continue`.

Basic Example Development Session

```
% source setup-ece2400.sh
% mkdir -p ${HOME}/ece2400/hi
% cd ${HOME}/ece2400/hi
% echo "#include <stdio.h>" > hi.c
% echo "int main() { printf(\"hi\n\"); }" >> hi.c
% gcc -Wall -o hi hi.c  # compile
% ./hi                  # run
% gdb -tui hi           # debug
```

Building, Testing, Debugging, Formatting, Profiling

```
% source setup-ece2400.sh
% mkdir -p ${HOME}/ece2400
% cd ${HOME}/ece2400
% git clone git@github.com:cornell-ece2400/netid
% cd netid
% TOPDIR=${PWD}

% mkdir -p ${TOPDIR}/pa1-math/build
% cd ${TOPDIR}/pa1-math/build
% cmake ..                     # generate makefile
% make check                   # run all test progs
% make check-milestone         #  ... for milestone

% make sqrt-iter-basic-test    # build one test prog
% ./sqrt-iter-basic-test       # run all test cases
% ./sqrt-iter-basic-test 1     # run test case 1

% make memcheck                # check memory issues
% ece2400-valgrind ./sqrt-iter-eval 100
% make coverage                # gen code coverage
% firefox coverage-html/index.html
% make autoformat              # autoformat code

% mkdir -p ${TOPDIR}/pa1-math/build-eval
% cd ${TOPDIR}/pa1-math/build-eval
% cmake -DCMAKE_BUILD_TYPE=eval ..
% make eval                    # build all eval progs
% make sqrt-iter-eval          # build eval prog
% ./sqrt-iter-eval 100         # run eval prog

% perf record --call-graph dwarf ./sqrt-iter-eval 100
% perf script report stackcollapse \
   | flamegraph.pl > graph.svg
```

ECE 2400 Linux, Git, C/C++ Cheat Sheet

Example C Program

```c
#include <stdio.h>

int avg( int x, int y )
{
  int sum = x + y;
  return sum / 2;
}

int main( void )
{
  int a = 10;
  int b = 20;
  int c = avg( a, b );
  printf( "avg of %d and %d is %d\n", a, b, c );
  return  0;
}
```

Coding Conventions

- Try to keep lines less than 74–80 chars
- Include header comment at top of each file
- Include comments to explain code
- Only use // comment style
- Use only spaces, no tabs; use two-space indentation
- Avoid two blank lines in a row
- Use horizontal whitespace to separate conceptual things
- Use `CamelCase` for class names
- Use `under_scores` for variable names
- Use informative class/variable names
- Declare variables close to first use of variable
- Do not declare multiple variables in single stmt
- Place * with type not variable name (`int* a;`)
- Open curly brace on next line for function defintions
- Open curly brace on same line for conditional stmts
- Open curly brace on same line for iteration stmts
- Avoid global variables

Bad Style

```c
double foo= a;
int b =bar;
double c=1;
double d,e;
int *f_p = &b;
```

Good Style

```c
double foo = a;
int    b   = bar;
double c   = 1;
double d;
double e;
int*   f_p = &b;
```

Example C Header File

```c
#ifndef AVG_H
#define AVG_H

int avg( int x, int y );

#endif AVG_H
```

Example C Source File

```c
#include "avg.h"

int avg( int x, int y )
{
  int sum = x + y;
  return sum / 2;
}
```

Example Test Program

```c
#include "avg.h"
#include "ece2400-stdlib.h"
#include <stdlib.h>

void test_case_1_even()
{
  printf("%s\n", __func__  );
  ECE2400_CHECK_INT_EQ( avg( 2, 4 ), 3 );
  ECE2400_CHECK_INT_EQ( avg( 3, 7 ), 5 );
}

void test_case_2_uneven()
{
  printf("%s\n", __func__  );
  ECE2400_CHECK_INT_EQ( avg( 2, 5 ), 3 );
  ECE2400_CHECK_INT_EQ( avg( 3, 8 ), 5 );
}

int main( int argc, char* argv[] )
{
  __n = ( argc == 1 ) ? 0 : atoi( argv[1] );
  if ( (__n <= 0) || (__n == 1) )
    test_case_1_even();
  if ( (__n <= 0) || (__n == 2) )
    test_case_2_uneven();
  printf( "\n" );
  return __failed;
}
```

Example C++ Program

```cpp
class Point
{
 public:

  // Default constructor
  Point() : m_x(0.0), m_y(0.0) { }

  // Non-default constructor
  Point( double x, double y ) : m_x(x), m_y(y) { }

  // Accessors
  double get_x() const { return m_x; }
  double get_y() const { return m_y; }

  // Member function
  void translate( double x_off, double y_off )
  {
    m_x += x_off;
    m_y += y_off;
  }

 // Private data members
 private:
  double m_x;
  double m_y;
};

// Overloaded operator
Point operator+( const Point& pt0,
                 const Point& pt1 )
{
  Point tmp = pt0;
  tmp.translate( pt1.get_x(), pt1.get_y() );
  return tmp;
}

int main( void )
{
  Point pt0(1.5,2.5);
  Point pt1(2.5,3.5);
  Point pt2 = pt0 + pt1;
  return 0;
}
```