Introduction

"Welcome to Terminal IDE.

Terminal IDE is an expandable command-line based system, with many useful applications, that runs completely within this 'Application's Context', so you don't need ROOT or
any other special permissions. The whole application works under the normal Android permissions. :-)

Log in via telnet or ssh and use your nice big home keyboard and computer screen for maximum-super-power!
utelnetd and sshd are available and the preferred method of using this app.

You also have a complete 'java' development environment, for creating java command line apps or full Android applications.
A collection of sample applications that show how to build and run apps is available in ~/system/src/

bash, vim, javac, java, dx, aapt, apkbuilder, signer and many many more, thanks to busybox, are all available from the provided command line.

Since Android is not designed to be used with an old fashioned ANSI keyboard, with CTRL ESC ALT and the rest of them, a complete ANSI keyboard has been created.

By using the 'Terminal IDE keyboard', which you will have to enable in the Settings > Language and Keyboards of your device, you
will get a fully working soft-keyboard with TAB, CTRL, ALT, DEL, Function keys etc.. AND all the keys work as they should. Only really to be used with Terminhal IDE.

One other IMPORTANT issue is hostname resolution. Due to a mixture of statically linked C binaries and no /etc folder on un-rooted phones,
hostname resolution generally doesn't work in Terminal IDE. BUT this only applies to C apps. Java is fine :-).
Use 'jping' to get the IP of any hostname you need to use and then use that 'numeric' IP instead of the 'string' name and the C app (wget, git, mc ftp access ..)
should work fine, ie try..

```
# jping www.google.com
```

And it will tell you google's IP.

Also - SSL is not built-in to most of the C apos. You do have ssh of course and can create SSL proxy pipes though that (tutorial coming some day.. it's easy).

For the most amount of help possible, use Google! There are an infinite amount of tutorials on vim, bash, busybox and all the other available apps.

Make sure you READ, FULLY understand, and can PERFORM, the 'Tutorials' in FULL. This will show many features..

So. The road ahead is long and hard. But the rewards are great. That's why it's long. And hard.

Enjoy

Keyboard

The Keyboard is VERY important. You need access to the CONTROL keys of a keyboard or none of these command line applications will work properly..

Terminal IDE comes with a full 128 set ASCII soft 'on screen' keyboard.

You will need to enable it in the Settings > 'Language and Keyboard' section of your device.

You can select the keyboard from the main screen by pressing 'Keyboard'. You can switch back to your regular keyboard easily by clicking the options button in the bottom right of the keyboard itself.

There is a small-ish version and a larger version, and you can set them differently depending on the orientation, in the Keyboard Settings.

Terminal IDE also comes with a powerful Hard Keybaord Mapper so that MOST external keyboards should now work. And work well.

As a basic setup, you can specify a CTRL key and ESC key in the options, so atleast you will always have them.

CTRL is the only TOTALLY essential one.. as..

ESC   = CTRL+[
[ALT+X] = [ESC then X] (Basically..)

To see the key combo in BASH pres CTRL+v and then try an action. It will display what keys it receives.

To set your hard (bluetooth etc..) keyboard up correctly you need to use the 'Hard Key Mappings' section in the 'Options'.

Before you do - check what does / doesn't work, as most things now should work automatically.

Function Keys are a usual problem, so you will have to pick a key on the keyboard you can use as the Function key modifier.

First - in the main terminal view, press Menu, and select Toggle Key Logger. This is ALWAYS off by default. This will output the keys you press, on your external keyboard, to the file ~/.keylog

Then from the command line either type

```
# tail -f ~/.keylog
```

(To exit press CTRL+C)

This will poll the 'tail' end from that file and output it to std out. It refreshes every half second so expect a slight delay. Now press any key, and that 'key code' is the value you need to use in the Hard Keyboard Mappings section.

Or the KEY LOGGER output is also sent to ADB logcat. So you can check it there too.

If you press a key and no key-code is sent, like those little blue FN keys, they only change the keyboard.. They cannot be used. They must send a valid code to be used.

You can map ANY key from your keyboard to ANY key in the list. As long as it sends a key code..

Don't forget to Toggle the key logger OFF when you are finished.

At the top of the Hard Mapping Section you can enable or disable this feature, in case you override some system keys, and if you want to reset the key values just set them to -1. Only INTEGER values allowed.

My Sony bluetooth keyboard now works 'perfect' with this method..

Tutorial 1

"Follow all these tutorials through EXACTLY.

They will help you compile the sample applications in ~/system/src step by step.

The best way to follow these tutorials is by logging in over telnet/ssh and then
you can read and follow on the device, whilst using the computer to perform the actions.

Let's Begin..

Installing the basic system.

Click \'Install System\' on the main page, and then follow the instructions.

Make sure you have enabled the Terminal IDE Keyboard in Settings > Langauge and Keyboard on your device. Then choose the Terminal IDE keyboard - click \'Keyboard\' on the main page.

Start Terminal IDE. Click the button..

You will now be presented with a nice not-short color prompt. You can change this by typing,

```
..$ export PS1=#
```

But it does look really nice over telnet/ssh.. :-p

- Double-tap toggles the keyboard.
- Swipe left and right to view your 4 Terminals.
- Long pressing the screen brings up the Terminal Chooser menu.
- Press \'Menu\' for Options

[This part is optional but highly recommended]

Run telnetd. No parameters are rquired. Defaults setup in ~/.bashrc.You
can see what is actually being run by typing :

```
# cat `which telnetd`
```

Anyway - start telnetd with :

```
# telnetd
```

You should now see telnetd startup info. Ending in ../system/bin/bash

You can shut telnetd down at any stage by typing [CTRL]-C on the Terminal IDE Keyboard

Now, go to your nice big computer and login via telnet. I have set the escape character to blank, as it is the same as the vim help link follow key.

[Either]

```
$ telnet -e '' [THE PHONES IP] 8080
```

[Or - better still with adb, which is part of the Android SDK]
Plug your phone into computer via USB first, then

```
$ adb -d forward tcp:8080 tcp 8080
$ telnet -e '' 127.0.01 8080
```

['' is 2 single quotes not a double quote]

Now you should be connected to the system via TELNET!

[And you can connect multiple times simultaneously]

OK - let's start programming.

Tutorial 2

Let's start by compiling the simplest Hello World app.

First cd into the project folder (Remember to use TAB to complete) :

```
# cd ~/system/src/helloworld/
```

List the contents :

```
# ls (Allthough I prefer ll)
```

You will now see three files. hello.java is the code. Compile with :

```
# javac hello.java
```

Now you have a hello.class file. Need to convert to DEX format :

```
# dx --dex --output=hello.jar hello.class
```

Now you have hello.jar in dex format. The naming to .jar seems to matter, not .dex.

And finally you can run it with :

```
# java -jar hello.jar hello
```

You just compiled and ran your first program.. Well Done!

Now you will understand what builder.sh and run.sh are doing :

```
# cat hello.java
# cat builder.sh
# cat run.sh
```

Now let's play with vim a little.. VIM not VI - they are different! vi is included only as a fail-safe, from busybox, in-case you break vim with 'weird' config settings.

```
# vim hello.java
```

You should now see a syntax highlighted hello.java file.

REMEMBER : The most important thing about vim is that [ESC] moves you into Control mode where you can type commands and do functions, and press \'i\' for INSERT mode to write some text.(Or use \'a\' for APPEND mode)

Press \'i\' to enter INSERT mode

Move around with the arrow keys, and then move over the \'System.\' so that the cursor is on the \'o\' of
.out

Now press [CTRL]+X [CTRL]+U

You should see java completion for the available options!

There are many available completion type :
- [CTRL]+X [CTRL]+U   USER java completion
- [CTRL]+X [CTRL]+O   OMNI completion, syntax based for many languages
- [CTRL]+X [CTRL]+F   FILE name completion, VERY HANDY, try ~[CTRL]+X [CTRL]+F

And more.. (Enable the dictionary in your ~/.vimrc for many more options) But for us the important one is the USER completion. Java.

Completion only works in INSERT mode.. The auto-popup vim plugin is included, so if you type normally, after the 2nd letter of a word,   the auto-completion window should just popup automatically.. sweet. :-)

You can even compile the code with [F7] (this is setup in your ~/.vimrc).

Make sure you are in Control mode, press [ESC], then press [F7].

Since there are no errors the compile will succeed and no errors will appear in the
output window that should now have appeared at the bottom of the screen.

The Cursor is now in the output window. Move it to the main window.

Press [ESC] (to make sure you are in control mode). Then press [CTRL]+W (to move into window mode) and then press the UP arrow key.
This will move you to the window \'above\'. You can use [CTRL]+W [CTRL]+W to toggle between windows.

You can close the output window if you wish :

[ESC]:cclose[ENTER]

Bring it back with :copen or :clist
Cycle with :cnext :cprevious

Edit the hello.java file by entering INSERT mode and changing it so that there are a few errors.. add a space inbetween the letters of System etc..

You will need to save your changes. Make sure the cursor is in the main window. Then :

[ESC]:w

[ESC] is only required if you are not in Control mode.. From now on I'll omit it. :w is save the file. Most functions
need to be run from Control mode, and all those starting with :

Now try and compile again with [F7]. You need to be in the main window in Control mode.

This time there should be some errors in the output window.

You can cycle through them with [F8] and [F9], and the cursor will move to the error line.

Pretty sweet..

You can even run your shell build scripts from within vim, so..

```
:! ./builder.sh
```

This will run the complete build script, and show you the output.

NB : You can use TAB completion in the vim command line so just type :

:! ./b[TAB] and the rest should be filled in..

To exit vim use :

```
:w    - Save file
:q    - Close but file must be saved
:q!   - Force Close
:wq   - Save and Close
:qa! - Force close ALL.
```

Ok. Play around here, then move on to Tutorial 3.

Tutorial 3

We will now compile a more substantial java library. This library will the be used in the 2 last remaing apps.

Android requires that classes be in the DEX format when running them, but javac requires them in the regular .jar format

This project compiles both types.

First cd into the project home folder

```
# cd ~/system/src/demo_lib/
```

Have a look around

```
# ls
```

Check out the builder script

```
# cat builder.sh
```

This will explain all that is happening. Two libs are created at the end.

Run the build script

```
# ./builder.sh
```

To check the results run

```
# ls dist/
```

Ok - now with vim

Rather than start vim from this folder, it is important to start it from the src/ folder, as this way javac is called with the right filenames when compiling.

I have written a simple script called 'terminalide' to help.

```
# cat `which terminalide`
```

This will show how simple a script it is. Run it.

```
# terminalide
```

vim will now start with a lovely file-browser side-pane on the left, setup in the correct home folder.

This is done using a vim plugin. :NERDTree

Move to the src dir with the arrow keys, and press [ENTER] to open a directory or file.

Open the libfunc.java file in ./src/org/library by movimg onto it and pressing [ENTER]

The cursor will now be in the main window now. Press [F7] to compile.

At this point there are no errors. Make some errors in the code and re-compile.

Now cycle through your errors with [F8] and [F9].

I think you'll agree, this is starting to look like a proper IDE.

You may have noticed a file called builderpro.sh and proguard.cgf

This is the build script to create the lib using proguard obfuscation.. There are similar scripts in the other demo_ apps

Play around, then :qa! (exit) and go to Tutorial 4.

Tutorial 4

We will now compile a complete command line java application, that uses the library we compiled in Tutorial 3.

cd into the home folder

```
# cd ~/system/src/demo_console/
```

Check things out..

```
# cat ./builder.sh
# cat ./run.sh
```

Notice how the lib file has been integrated into the build commands.

Also, check out the terminalide.vim file

```
# cat ./terminalide.vim
```

Just take a look at it. It will make sense later.

Now run terminalide

```
# terminalide
```

Open the start.java file in the src dir.

Compile with [F7]..

There are ERRORS! This is normal.

Also, the java-complete functions will not work for libfunc.

This is because the vim variables have not been told about the new build environment and added libs.

Run this function in vim. Make sure you are in the main window and in Control mode.

```
:source ../terminalide.vim
```

This will set the variables up correctly for the window so that java-complete and javac now work.

Now try and compile. It should work fine. And the java completion for libfunc aswell..

Use the builder script to create the full app.

```
:! ../builder.sh
```

Now lets open a second file.

Move to NERDTree

```
[ESC][CTRL]+W LEFT
```

Open up name.java aswell. You will now notice a thin window appear at the top with the names of your current buffers. Also name.java will now be visible in the main window

Move into this MiniBufExpl window, [CTRL]+W UP

Press TAB or SHIFT-TAB to cycle through the windows, and ENTER to select one. Nice.

There are other nice ways to switch buffers.

:buffers

this shows a list of buffers, and you then choose with

:buffer [number]

I have created a simple mapping in ~/.vimrc so just press [F5]..

Play around here and then go to tutorial 5.

Tutorial 5

You should now be pretty adept at compiling code and editing in vim.

Here is a complete Android application.

cd into the home folder

```
# cd ~/system/src/demo_android/
```

Check out the build script

```
# cat ./builder.sh
```

Quite a beast.

But once you understand it, you will see that the eventual result is the creation of an .apk file.
This is signed with a default test signature.

This will show you how the install is performed

```
# cat ./install.sh
```

And then you can install it..so

```
# ./builder.sh
# ./install.sh
```

And run it..

# ./run.sh

And HEY presto. If all's well, you should see an Android application start on the Device!

Now for vim..

Check out the terminalide.vim file first

# cat ./terminalide.vim

It is almost the same as the demo_console version but the classes are built in the ./build/classes folder instead.

Use terminalide to start vim

# terminalide

Open a the main java file. And then remember to setup correctly

:so ../terminalide.vim

Now javac compile and java-complete should work fine.

Open multiple files and switch between them using the MiniBufExpl window at the top.

It all works... finally.

The rest - is up to you.

----

bash

The command line. THIS is where all the real action takes place.

Once you are used to using the command line, nothing comes close to the speed and power it offers.

Here are some of bash's features

Edit your settings in the ~/.bashrc file :

# vim ~/.bashrc

UP / DOWN - shows your history. Very handy.

Tab Completion - single TAB to complete or double TAB to show all the available options.

Change your prompt with > export PS1=#

Ctrl-c CANCEL current process

Search for the last time you used a function..
Ctrl-r REVERSE search

Show the keys actually being sent to bash
Ctrl-v VIEW next key combo

Undo all changes to the current line. (When you scroll up through the histroy, if you make changes)
Alt-r REVERT to original line

Some other shortcuts :-

Ctrl-l	Clear the screen leaving the current line at the top of the screen.
Ctrl-u	Delete from the cursor to the beginning of the line.
Ctrl-k	Delete from the cursor to the end of the line.
Ctrl-w	Delete from the cursor to the start of the word.
Ctrl-b	Move back one character.
Alt-b	Move back one word.
Ctrl-a	Move to the start of the line.
Ctrl-e	Move to the end of the line.
Ctrl-f	Move forward one character.
Alt-f	Move forward one word.
Alt-] x	Where x is any character, moves the cursor forward to the next occurance of x.
Ctrl-y	Pastes text from the clipboard.

---

busybox

The swiss army knife of applications.

99% of the apps available in Terminal IDE are there because of busybox.

An App that allow you to compress
```
tar -czf compressed.tar.gz ./folder-to-compress
```

And extract
```
tar -xf compressed.tar.gz
```

Or list the contents
```
tar -tf compressed.tar.gz
```

If you are recursively looking for a file, based on the name
```
find . | grep search_term
```

If you are recursively looking for a file, based on the contents
```
grep -ar "searchforthis" *
```

And many many other apps far too numerous to mention.

Use them wisely..

## vim

There is only one number bigger than infinity. The number of tutorials for vim
on Google. This will be your best source of information.

vim ~/.vimrc : to check your current settings.

One IMPORTANT feature is using the command line app \'terminalide\' to start vim. This is only really helpful if you are working on a java project. run \'cat `which terminalide`\' to see the script in full.

This is a simple but useful script that makes sure you start vim in the correct folder, so that javac can work correctly. All
that is required is that there is a folder named \'src\' in the current folder.

Run \'terminalide\' from the project home folder. vim will start with a nice file tree (NERDTree)that you can use to browse for a file.

Basically, make sure you start vim in the src/ folder of any java projects. Then javac works fine.

The basics have been setup in the initial ~/.vimrc file and ~/.vim folder. You have working arrow keys, backspace, delete, Page UP/Down etc. All \'NOT\' givens in vim-land, I might add..

Vim has been setup with java in mind, and so has all the features required to be a powerful IDE, including java class / function popup completion..

The plugins that you have in ~/.vim are
[Search google for their full uses..]
: NERDTree - a nice file browser
: SnipMate - auto text copy. Check snippets in ~/.vim and use word[TAB] to activate.
: MiniBufExpl - the buffer window at top of vim that shows multiple buffers.
: ACP - Auto Complete Popup, you can disable this by removing acp.vim from your plugins folder.
: Java-Complete - hacked/fixed up and working great!

Add your own plugins to ~/.vim and edit the ~/.vimrc file to gain total control.

# vim ~/.vimrc

Read / Edit the settings file to understand more..

----

javac, java, dx..

Terminal IDE comes with a complete Java / Android development kit, and has a very powerful and java optimised editor - vim.

A set of sample applications can be found in ~/system/src. I strongly recommend you read the Tutorialhelp through to the end. It will show you what you need to know.

vim has been specifically configured for java!

Know that these features are supported :-

-Full syntax highlighting - of numerous languages
-Auto popup completion of files, words and syntax of numerous languages
-Java-Completion - in vim insert mode , CTRL-X CTRL-U
-One button Compile [F7] - check ~/.vimrc
-List of errors on compilation and ability to jump through them [F8][[F9]
-Left pane file-browser - the vim NERDTree plugin
-Lots more..

Compiling for Android is a little different, as after the normal javac compile, classes need to be converted to the \'dex\' format Android\'s dalvikVM requires.

Also - \'aapt\' and \'apkbuilder\' are required to build the resources and final .apk file.

All of this is provided..

Android has a Java VM but it is called Dalvik. It needs to be called in a special way to enable you to run java applications from the command line. (Especially as a non-root user!)
All this has been fixed and now works. Voila.

Review the build scripts used in the samples to see how all the different apps are pieced together.

---

telnet

This is the MOST important feature of Terminal IDE.

Being able to log in over telnet / ssh with a large fully compatible keyboard and even larger monitor is what makes Terminal IDE so useable.

<u>How to login via TELNET</u>

A typical session will entail you starting Terminal IDE and simply running 'telnetd'. No options required as defaults are used.

Then log in via your secure home wifi with telnet, default port 8080.If you are an android developer and have adb installed, there is a better option.

Plug your phone in via usb and make sure adb can see it. Then run adb -d forward tcp:8080 tcp:8080

This will forward the local port 8080 to the device port 8080

Then log in via telnet, via your local machine. Simple.

telnet -e '' 127.0.0.1 8080

Since there is no encryption, like ssh, and over USB, this is the fastest possible connection..

Using the -e '' turns off the escape character, which is the same as the default vim help-link follow-through key. Set it to whatever you want.

Once you have logged on make sure the TERM variable is correct, The default is 'xterm'.

----

ssh

How to login via SSH

There is a slight issue.. basically when you log in you have to start bash manually.. unless you have the file /etc/shell with the correct shell to use.. Which requires a rooted phone.

Since Terminal IDE is for non-ROOT users, I will have to recompile the code to allow a shell to be specified on the command line.. Soon..

FOR NOW - This is how to connect to the phone via SSH (There are other ways using public keys but this is one way)

So - Once in Terminal IDE

2) You need to create a couple of server ssh keys

Start in $HOME
# cd ~

Create folder
# mkdir .ssh

Give it some secure permissions
# chmod 700 .ssh

Get in there
# cd .ssh

Now create the keys
# dropbearkey -t dss -f dropbear_dss_host_key
# dropbearkey -t rsa -f dropbear_rsa_host_key

ok - That's almost it. Just need to start dropbear with the correct parameters now. [Probably want to keep this in a script]

Back HOME

```
# cd ~
```

You need to know the UID of your app, which is different per phone - use 'id'

```
# id
```

That will tell you your user ID / Group ID. Let's say its 10058.

Now to start DropBear

```
# dropbear -A -N username -U 10058 -G 10058 -C password -d ~/.ssh/dropbear_dss_host_key
-r ~/.ssh/dropbear_rsa_host_key -F -E -p 8090 -P PidFile
```

You should DEFNITELY write a simple script to do this.. Call it sshd.. and put it in ~/bin

This will start sshd running in the foreground with password set to 'password' on port 8090.

Then you can connect, like telnet, and simply use 'password' for the password.

Now for the issue. It will start a simple shell session in / with no ENVIRONMENT variables or anything..

I'll fix it permanently in a future release, but for now it can be fixed with these 2 commands.

cd into your home dir - Check this is correct on your device

```
# cd /data/data/com.spartacusrex.spartacuside/files
```

And start bash with an init file Terminal IDE auto-magically creates..

```
# ./system/bin/bash --init-file ./.init
```

Everything should now be setup as usual.

You can set up a nice script that will do this for you automatically on startup..

You need to invoke ssh like this..

```
# ssh 127.0.0.1 -p 8090 -l username -t
/data/data/com.spartacusrex.spartacuside/files/system/bin/bash --init-file
/data/data/com.spartacusrex.spartacuside/files/.init
```

Good luck..

----

rsync

rsync is a great way to backup and synchronise folders on   different machines.

It only sends the DIFFERENCE between files so that the minimum amount of bandwith is used.

Even large projects can easily be synced quickly, if you're only transferring the changes..

There are MANY ways of syncing folders in rsync, pull or push, run in daemon mode, etc.. but I will show you ONE secure way of doing this.

You need to have the sshd daemon running in Terminal IDE. Follow the previous instructions.

Now its really quite simple. You need rsync on your home computer.

Let's try and synchronise the ~/system/src demo apps folder.

On your big computer, where you want to backup your work, you need to run this function :

```
# rsync -e 'ssh -p 8090'
--rsync-path='/data/data/com.spartacusrex.spartacuside/files/system/bin/rsync' -avn
127.0.0.1:/data/data/com.spartacusrex.spartacuside/files/system/src .
```

By adding the -n option it will show you what it would do, but not do it. Re-run without the -n to actually back the files up..

```
# rsync -e 'ssh -p 8090'
--rsync-path='/data/data/com.spartacusrex.spartacuside/files/system/bin/rsync' -av
127.0.0.1:/data/data/com.spartacusrex.spartacuside/files/system/src .
```

Create a script, to make this easier to call.

And there you have it. Fast simple secure folder synchronization..

You can of course reverse the rolls of this script.

That would mean you could backup your files FROM Terminal IDE TO any ssh server you have access to online, as long as it has rsync.

GIT

GIT 1.7.8

[ISSUE : The hostnames do not resolve correctly so you must use the NUMBER IP format for any git servers you use.

github.com = 207.97.227.239]

Git is a fast, open source, distributed version control system.

It is a simple command line tool for keeping a history on the state of your source code projects. You use it as you might use something like Subversion, CVS or Perforce.

You tell it to track files in your project and periodically commit the state of the project when you want a saved point. Then you can share that history with other developers for collaboration, merge between their work and yours, and compare or revert to previous versions of the project or individual files.

Git is fully distributed, which means that it can work almost entirely offline. In stark contrast to VCS tools like Perforce or Subversion, Git does nearly all of itâ€™s operations without needing a network connection, including history viewing, difference viewing and commiting.

Probably the most compelling feature of Git, since it often fundamentally changes the way that many developers work, is Gits branching model. Instead of the popular VCS branching method of simply cloning into a seperate directory for a branch, Git lets you switch between branches in a single working directory. Add to that the fact that creating and switching between branches is nearly instant, not all of your branches need to be shared, and itâ€™s easy to stash partially completed work - means that the way you work can be incredibly different.

Instead of only having branches for major development line departures, Git developers routinely create, merge and destroy multiple branches a week, or even per day. Often each feature or bug you are working on can have its own branch, merged in only when it is complete. This model allows you to experiment quickly, easily and safely - without having to go through hoops to get back to where you where. It enables and encourages a non-linear development cycle, where you can work on multiple lines of thought in parallel without them stepping on each other.

To Start. Git records your name and email address when you create commits, so you need to tell Git what those are. You can use the git config command to set those. If you pass --global, it will save the values in the â€˜~/.gitconfigâ€™ file so they are the default for all of your repositories.

```
# git config --global user.name &quot;Scooby Dooby&quot;
# git config --global user.email &quot;woof@scoobymail.com&quot;

# git config user.name
# git config user.email
# cat ~/.gitconfig
```

Will show settings

To initialize a Git repository from an existing directory, simply type git init while in that directory. That will create the skeleton of the basic Git repository for you in that directory.

```
# cd ~/system/src/demo_android
# git init
# git add .
# git commit -m 'The Initial Commit'
```

You now have a working 'gitted' project that you can clone, branch, share and use..

```
# git log
```

You can exit the logger by pressing 'q'

If you now go to another directory you can clone the project easily.

```
# cd ~/projects
# git clone ~/system/src/demo_android
```

This obviously works for online git servers aswell.. [BUT Only use number IPs as hostnames don't resolve yet.. no /etc/hosts..]

Now lets make some changes..

```
# cd demo_android
```

Now edit something..

```
# vim terminalide.vim
```

Add a line comment and save the file [like &quot;This is an addition]

You can see your changes by running

```
# git diff
```

And commit them with

```
# git commit -a -m &quot;Some stuff about the update..&quot;
# git log
# git status
```

The status tells you what branch you are working on..

Or you can use

```
# git branch
```

To create a new branch, we can use 'git branch (branchname)' which will create a branch at the point we're currently at.

```
# git branch experiment
```

To switch to that branch so that the work we do is saved to it instead of the â€˜masterâ€™ branch, we run the â€˜git checkoutâ€™ commandâ€™

```
# git checkout experiment
# git branch
```

Now lets make more changes to the terminalide.vim file. Open it up and add some more text, that is commented out..

And commit those..

```
# git commit -a -m &quot;Edited terminalide.vim&quot;
# cat terminalide.vim
```

Now - you have 2 valid branches. If you want to switch back and check the master stuff, simply do

```
# git checkout master
# cat terminalide.vim
```

And switch back with

```
# git checkout experiment
# cat terminalide.vim
```

You can add files to the branch with

```
# git add [file]
```

Once happy with your branch you can merge it back to the master.

```
# git merge experiment
```

Once you have done all your changes and merging, and have finished with your branch you can delete it with

```
# git branch -d experiment
```

And Voila! That was a nice little git session. There is of course a lot more that can be done with git, so, as always, GOOGLE IT!

Midnight Commander

Midnight Commander 4.8

Just type 'mc'.

This is a great console based file manager.

Use the arrow keys to move around, TAB switches panels, and function keys to get the menus.

If the function keys don't work for you, just use [ESC] 'number' for the same effect.

You can mount FTP file systems in mc and if you set up an SSL pipe with ssh, you can access ftps.

There is even the FISH protocol for SSH file access.

To get it to work, I had to write a simple little mc script. mc-real is the actual binary. You can run mc-real on a regular screen / keyboard.

Few tips:

[F1] HELP - doesn't work yet :-(
[F3] View a file - v quick.
[F4] Open the file in vim for editing
[F9] gets you the main menus. Just use arrow keys and enter.

Press [ESC] twice to cancel most actions.

[F10] Quit.

As with vim, plenty of help on Google.. :-)

TMUX

TMUX 1.5

Tmux is a 'Terminal MultiPlexer'. Like GNU Screen but better.

What this means is that you can now have multiple windows arranged however you like in the SAME SINGLE console window.. :-)

A word of WARNING! If you've never used a feature like this, it may seem a little strange. Once you have used a feature like this, you can NEVER go back.. that's how utterly awesome it is..

Now - lets take her for a spin.

The tmux settings are in ~/.tmux.conf. Check them out. I have made a few shortcuts but very little else. That will be up to you.

```
# tmux
```

[If you are running this on the device screen itself you may want to start with

```
# tmux -u
```

as this will render the lines correctly..]

You should now be seeing a normal command line, and a status bar of information at the bottom of the screen.

Let's split the window [Very much like in vim..]

[CTRL]+a |

And again, this time vertically

[CTRL}+a -

Now you should have 3 panes open. And you will be in the bottom right one.

```
# top
```

Or even better..

```
# htop
```

And now switch to the next pane with

[CTRL]+a o

or even easier..

[CTRL]+a [Arrow Key]

Well - not bad, you now have 3 windows with a constant computer monitor..

Now lets create a whole new window..

[CTRL]+a c

You should now see 2 windows represented at the bottom.

In this new window why not start Midnight Commander

```
# mc
```

You can switch between the last active window by using

[CTRL]+a [CTRL]+a

Or

[CTRL]+a [n or p]

You can have as many windows and panes as our squishy biological brains can handle.

Imagine the possibilities.. tail -fs.. etc.. :-)

A list of some useful functions :

```
[CTRL]+a c   > Create new window
[CTRL]+a a   > Move to last used window
```

[CTRL]+a n   > Move to next window
[CTRL]+a p   > Move to previous window
[CTRL]+a [Window Number]   > Move to the specified window

[CTRL]+a | > Split window horizontally
[CTRL]+a -   > Split window vertically
[CTRL]+a o   > Move to next pane
[CTRL]+a [Arrow Key] > Move to pane in direction of arrow key
[CTRL]+a q   > Show pane numbers, and press quickly to jump

[CTRL]+a ,   > Rename current window
[CTRL]+a ?   > List all keybindings
[CTRL]+a [> Scroll Mode [Enter exits]
[CTRL]+a w   > List windows

You need to 'exit' and shut down the window panes individually. When all are shut down, tmux stops.. There may be a faster way?

There is more info about screen resizing, pane switching , detaching, re-attaching and MORE, but as always GOOGLE it.. there is plenty out there.

## BitchX

BitchX final 1.1

This is an IRC client. This stands for Internet Relay Chat.

If you are stuck on a problem, you can now ask for help in an infinite number of IRC chat rooms..!

You can set your IRCNAME and IRCNICK environment variables to speed things up.

ISSUE :-(.. BitchX works fine over SSH / Telnet but when run on the device the screen rendering has some issues. To get around this, when running on the device, I start it with

# BitchX -d (This will set 'dumb' terminal.. only ONE chat window allowed [I think?].. but it works)

[You can of course run multiple copies of BitchX in TMUX..]

Normally To start just run

# BitchX

Which will connect to your defaults.. or

# BitchX -n [nickname] [irc-server]

Terminal IDE does not resolve hostnames, as there is no /etc/resolv.conf on un-rooted phones, so the servers MUST be entered in their IP NUMBER format.

Once in BitchX.. some functions..

/list - Lists all the Channels
/quit - Exit BitchX
/window [func] - many functions - GOOGLE IT..
/window new - New window
/window kill - Kill window
/window goto # - Goto window number #
/join [channel] - Join a channel in the active window
/part [channel] - Leave a channel.

You can do all the window splitting and tabbing that you can with other good command line apps.

Chats away!

Compiling C Apps

Terminal IDE uses the command line with many native C apps. There are DEFINITELY more apps that some of you might want.

You can of course add C applications you have compiled yourself to Terminal IDE. Just put them in the ~/bin folder.. like your Java / Shell scripts.

No problem.. easy.. right?.. WRONG!! Well, it's not easy anyway..

Welcome to :

'The Dark Arts of Android Cross-Compilation..'

Anyone who has tried cross-compiling for android will know of what I speak..

The main issues that arise are :
1) ROOTED users may operate differently, but you need to set the application up, and all of it's configuration files, so that it runs from inside Terminal IDE's non-rooted memory space ie no /etc folder. This may involve $HOME config files, ENVIRONMENT variables, compiling the app with HARD-CODED path names, to mention a few.
2) In my experience, STATICALLY linking the apps greatly improves their ability to run on many different android devices. If you ONLY want it for your device, you can target it more precisely and build a dynamically linked binary.

'BUT dynamically linked binaries don't work on android', I hear you cry..

The reason dynamically linked binaries don't work is because Android doesn't follow Unix file system conventions. The dynamic linker is named /system/bin/linker, not /bin/ld.so as usual.

So you just have to tell your ld about it:
For an imaginary hello.c file

```
arm-none-linux-gnueabi-gcc -c -o hello.o hello.c
arm-none-linux-gnueabi-ld -rpath . hello.o libc.so -I /system/bin/ linker -o hello
```

Now you get a dynamically linked executable 'hello' which works on Android.

3) Even if you get the app to compile, there is still a chance it will just seg-fault when you run it on your device..
I'm looking at you 'svn'..and you 'qemu'.. :-(..plus, I have a fully working version of Bochs, but it is sooo sloowww..

The three build systems I have used to compile all the apps in Terminal IDE are :
1) Android NDK - the preferred option really, but the least compatible. I use the create-standalone-toolchain option mainly, but 'vim' is actually compiled using a proper Android.mk NDK project. It's the only one..
2) CodeSourcery ARM toolchain - works well, but you MUST statically link your apps.
3) BUILDROOT - that's how I built mc and htop..

Normally - I try to compile with the first option and if that doesn't work, after hours/days/weeks of wrestling, I move to another system and see what happens there. Brutal, but true.

That's all I can say about that really. You'll encounter many problems on your various adventures, but I
will add, that when it finally compiles, runs, and works, that's a gooood feeling..

Good Luck!

[Pls let me know if anything working pops out..]

Trouble

All efforts have been made to ensure the smooth and correct running of this application.

If you find that Terminal IDE is behaving abnormaly though, there are 3 options :

1) Turn it off and run away. Not an option I would advise.

2) Write a harsh comment that says how this app is a pile of **** and you can't believe I even dared to waste your time.. Again, not a great option, but it does make me laugh when I read some of the stuff.. :-)

3) Send me a short email with phone type, android version, error type and any other information you think is relevant, and I'll fix it. Jackpot.

When I find a bug, I crush it. If I don't find it, and you do, and don't tell me, it lives and we all lose.. I'm not a mind-reader. Or a phone-reader. I'm not one of the X-Men.

I'm Spartacus Rex

Thanks

NONE of this is possible without the Open Source Community!.

'Who can fathom it's deepest thoughts..'

Terminal IDE is available under GPLv2.

Get the code from http://code.google.com/p/terminal-ide/

Thanks especially to :

: JackPal - for his excellent terminal emulator
: Tom Arn - for aapt, jarsigner
: DropBear - for sshd
: Bash Boys - for bash
: Busy Boys - for busybox
: Vim love - vim rocks
: Fang Cheng - for java-complete vim script
: Eric Lafortune - for proguard
: Java - who ever you are.

And many many more..

Simon Funk, Kevin Boone,..

Thank you.

GPL License

themselves, then this License, and its terms, do not apply to those
sections when you distribute them as separate works. But when you
distribute the same sections as part of a whole which is a work based
on the Program, the distribution of the whole must be on the terms of
this License, whose permissions for other licensees extend to the
entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest
your rights to work written entirely by you; rather, the intent is to
exercise the right to control the distribution of derivative or
collective works based on the Program.

In addition, mere aggregation of another work not based on the Program
with the Program (or with a work based on the Program) on a volume of
a storage or distribution medium does not bring the other work under
the scope of this License.

3. You may copy and distribute the Program (or a work based on it,
under Section 2) in object code or executable form under the terms of
Sections 1 and 2 above provided that you also do one of the following:

a) Accompany it with the complete corresponding machine-readable
source code, which must be distributed under the terms of Sections
1 and 2 above on a medium customarily used for software interchange; or,

b) Accompany it with a written offer, valid for at least three
years, to give any third party, for a charge no more than your
cost of physically performing source distribution, a complete
machine-readable copy of the corresponding source code, to be
distributed under the terms of Sections 1 and 2 above on a medium
customarily used for software interchange; or,

c) Accompany it with the information you received as to the offer
to distribute corresponding source code. (This alternative is
allowed only for noncommercial distribution and only if you
received the program in object code or executable form with such
an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for
making modifications to it. For an executable work, complete source
code means all the source code for all modules it contains, plus any
associated interface definition files, plus the scripts used to
control compilation and installation of the executable. However, as a
special exception, the source code distributed need not include
anything that is normally distributed (in either source or binary
form) with the major components (compiler, kernel, and so on) of the
operating system on which the executable runs, unless that component
itself accompanies the executable.

If distribution of executable or object code is made by offering
access to copy from a designated place, then offering equivalent