Development Tools: Makefile and Subversion

(Good tools for good programmers)

Emmanuel Fleury
<emmanuel.fleury@labri.fr>

LaBRI, Université de Bordeaux, France

September 1, 2017

Overview

1. Makefiles

2. Subversion: A Version Control System

Overview

1. Makefiles

2. Subversion: A Version Control System

Makefiles

Make is a **build automation tool** which helps to build executable programs and libraries from source code.

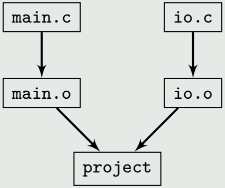
Coding the Dependencies

The user gives a dependency tree represented as a set of rules:

```
target :  dependencies
tab→ commands
tab→ commands
```

Example

```
project : main.o io.o
        gcc -o project main.o io.o

main.o : main.c
        gcc -c main.c

io.o : io.c
        gcc -c io.c
```



Few Tips (1/2)

Usual Targets

- `all`: Usually the first rule to build the software.
- `clean`: Clean unnecessary files.
- `help`: Display an help.
- `distclean`: Get back to a fresh distribution (no generated file).
- `.PHONY`: Is a special built-in target name. Set its dependencies as 'not a file', so that `make` will not try to generate the file.

Automatic Variables

- `$@`: Name of the target of the rule.
- `$<`: First prerequisite of the target.
- `$?`: All prerequisites that are newer than the target.
- `$^`: All prerequisites of the target (skip duplicates).
- `$+`: All prerequisites of the target (keep duplicates).
- `$|`: All the order-only prerequisites (ie. those which are not files).

Few Tips (2/2)

Recursive Calls

```
cd directory/ && make target
```

Generic Rules

The joker character is '%' and will be substituted by all matching files in the directory. For example, producing object files from C files is written this way:

```
%.o :  %.c
    gcc -c $<
```

Usual Variables

- `MAKE`: Path to the `make` software.
- `CC`: Path to the C compiler.
- `AR`: Path to the archiver.
- `CFLAGS`: Compiler flags.
- `CPPFLAGS`: Preprocessor flags.
- `LDFLAGS`: Linker flags.

A Complete Example

```
# Variables
EXE=project

# Usual compilation flags
CFLAGS=-std=c11 -Wall -Wextra -g -O2
CPPFLAGS=-I../include -DDEBUG
LDFLAGS=-lm

# Special rules and targets
.PHONY: all clean help

# Rules and targets
all: $(EXE)

$(EXE): main.o io.o
	$(CC) $(CFLAGS) -o $@ $^ $(LDFLAGS)

%.o: %.c
	$(CC) $(CFLAGS) $(CPPFLAGS) -c $<

clean:
	@rm -f *~ *.o $(EXE)

help:
	@echo "Usage:"
	@echo "␣␣make␣[all]\t\tBuild␣the␣software"
	@echo "␣␣make␣clean\t\tRemove␣all␣files␣generated␣by␣make"
	@echo "␣␣make␣help\t\tDisplay␣this␣help"
```

A Recursive Makefile

```
# Variables
EXE=project

# Special rules and targets
.PHONY: all build check clean help

# Rules and targets
all: build

build:
	@cd src && $(MAKE)
	@cp -f src/$(EXE) ./

check: build
	@cd test && $(MAKE)

clean:
	@cd src && $(MAKE) clean
	@cd test && $(MAKE) clean
	@rm -f $(EXE)

help:
	@echo "Usage:"
	@echo "␣␣make␣[all]\t\tBuild"
	@echo "␣␣make␣build\t\tBuild␣the␣software"
	@echo "␣␣make␣check\t\tRun␣all␣the␣tests"
	@echo "␣␣make␣clean\t\tRemove␣all␣files␣generated␣by␣make"
	@echo "␣␣make␣help\t\tDisplay␣this␣help"
```

Overview

1 Makefiles

2 Subversion: A Version Control System

Version Control System

Version Control System (VCS)

A Version Control System is a tool that:

- Keep a history of all changes applied on the source code.
- Allow to navigate within source code versions.
- Help to interact with other developers.
- Ease the creation and merge of branches within a source code.

Even when coding alone, a VCS is extremely useful to not be annoyed by tracking different revisions of your software.

Get used to always use a VCS on long term projects !!!

Subversion: A Modern VCS

Subversion History

The project started in **2000** aiming at being a **modern CVS**. Nowadays it is one of the most used centralized VCS in the Open-Source community. Though distributed VCS, such as Git, are gaining in strength and popularity.

Subversion: Command's and getting help

- Command: `svn <command> <options> <arguments>`
  Example: `svn commit -m "Added comments"`
- Listing all the basic commands: `svn help`
- Getting help on a command: `svn help <command>`
  Example: `svn help add`

Few Concepts and Words

Repositories

- **Main Repository**: Is the most up-to-date repository. Every developer must commit its modification on it. Usually it is a remote server.
- **Local Repository**/**Local Copy**: This is your copy of the main repository and it holds all your current modification of the source code. You need to synchronize it with the main repository from time to time.

Send/Get source to/from main repository

- `import`: Copy the content of a local directory to the main repository.
- `check-out`: Copy the source from the main repository to your local copy.
- `check-in`: Copy the source from your local copy to the main repository.

Usual SVN Module Structure

- `trunk`: The main branch of your project.
- `tags`: All the official releases of your project.
- `branches`: Experiments, trials or heavy changes that requires a separate space.

Create an SVN Module

1. Create the following hierarchy and populate the trunk directory with sources.

```
module_name/
     |-- branches/
     |-- tags/
     `-- trunk/*
```

2. Do: `svn import URL/module_name/trunk`
   (where URL is the address of the main repository)

3. Check-out the SVN module on your machine:
   `svn checkout URL/module_name/trunk local_name`

4. If the check-out went fine, just erase the initial directory and work in the checkout.

NEVER checkout something outside of the trunk directory !

(unless you know exactly what you do. . .)

Basic Usage

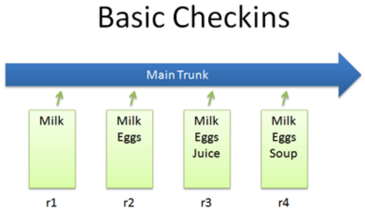
Commit or Check-in

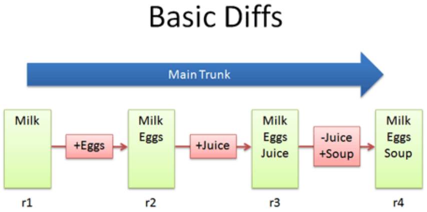
A **commit** or **check-in** saves a new version of the code into the main repository.
Before performing a commit, one can **modify**, **add** (`svn add`), **remove** (`svn rm`), **rename** (`svn rename`) a file or a directory. And then commit:
`svn commit -m "message"`

Diffs

A **diff** is a way to highlight differences between files. At any time while modifying the code you can see your modifications since last check-in with the command:
`svn diff`

```
Index: bar.c
========================================
--- bar.c (revision 3)
+++ bar.c (working copy)
@@ -1,7 +1,12 @@
 int main(void) {
-  printf("Four slices of Cheese.\n");
+  printf("Five slices of Cheese.\n");
 return 0;
```





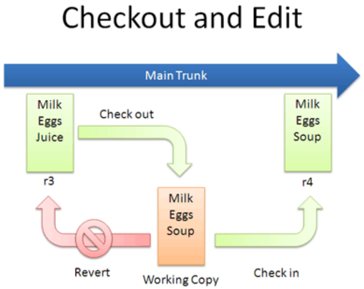
Syncing with Others

Usual Workflow

1. Checkout/update from the main trunk
2. Modify the source on your working copy
3. Commit (checkin) your changes

Few rules to avoid problems

- Never track files that are generated from others.
- Do not commit non-compiling code.
- Before committing:
  1. `svn status`
  2. `svn diff`
  3. `svn update`
  4. Solve conflicts
  5. `svn commit`
- Minimize your footprint on the code (do not introduce unnecessary modifications)
- Do not commit without a log message.



Useful Commands

- `add/delete`: Add/delete a file or a directory.
- `diff`: Display differences introduced in your local copy since last update.
- `commit`: Submit changes to the main repository.
- `mkdir`: Create a directory to be added to the main repository.
- `move/rename`: Move or rename a file or a directory.
- `revert`: Cancel local modifications of a file or a set of files.
- `update`: Synchronize your local copy with the main repository.
- `blame`: Track who did the last modification line by line.
- `info`: Display repository information.
- `log`: List commit messages which are in your local repository.
- `status`: List files and directories with a summary of their local modifications (see next slide).

Status List

```
  L     abc.c               # svn has a lock in .svn directory for abc.c
M       bar.c               # the content in bar.c has local modifications
 M      baz.c               # baz.c has property 'M' but no modification
X       3rd_party           # this dir is part of an externals definition
?       foo.o               # svn doesn't manage foo.o
!       some_dir            # svn manages it, but it's missing or incomplete
~       qux                 # versioned as file/dir/link, but type changed
I       .screenrc           # svn configured to ignore it
A  +    moved_dir           # added with history of where it came from
M  +    moved_dir/README    # added with history and has local modifications
D       stuff/fish.c        # this file is scheduled for deletion
A       stuff/loot/bloo.h   # this file is scheduled for addition
C       stuff/loot/lump.c   # this file has conflicts from an update
R       xyz.c               # this file is scheduled for replacement
    S   stuff/squawk        # this file or dir has been switched to a branch
```

Conflicts

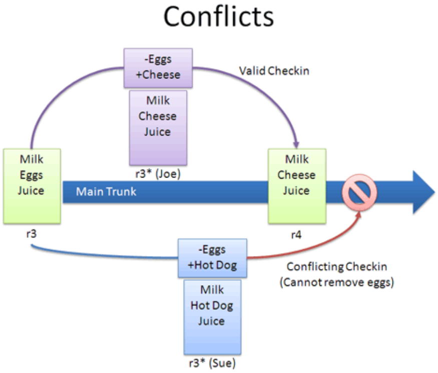
How a conflict works

When a conflict occurs, Subversion locks the local repository and you cannot check-in your changes to the main repository. During an `update`, conflicts are displayed by:

```
$> svn update
C  list.txt
Updated to revision 2.
```

For each conflict, three new files are created:

- `list.txt`: All the safe merges have been performed in this file. Unsafe ones are tagged inside the file.
- `list.txt.mine`: File as before `update`.
- `list.txt.r1`: File as before you modified it locally.
- `list.txt.r2`: File as it currently is on the main repository.



Important Commands

- `resolved`: Remove the '*conflict*' label from the file or directory.
- `cleanup`: Recursively clean-up the local repository, removing conflict labels and locks.

Example of Conflict Resolution

list.txt after a conflict

```
Top piece of bread
Mayonnaise
Lettuce
Tomato
Provolone
<<<<<<< .mine
Salami
Mortadella
Prosciutto
=======
Sauerkraut
Grilled Chicken
>>>>>>> .r2
Creole Mustard
Bottom piece of bread
```

Example of Conflict Resolution

list.txt after a conflict

```
Top piece of bread
Mayonnaise
Lettuce
Tomato
Provolone
<<<<<<< .mine
Salami
Mortadella
Prosciutto
=======
Sauerkraut
Grilled Chicken
>>>>>>> .r2
Creole Mustard
Bottom piece of bread
```

A possible merge of list.txt

```
Top piece of bread
Mayonnaise
Lettuce
Tomato
Provolone
Salami
Mortadella
Prosciutto
Sauerkraut
Grilled Chicken
Creole Mustard
Bottom piece of bread
```

Example of Conflict Resolution

`list.txt` after a conflict

```
Top piece of bread
Mayonnaise
Lettuce
Tomato
Provolone
<<<<<<< .mine
Salami
Mortadella
Prosciutto
=======
Sauerkraut
Grilled Chicken
>>>>>>> .r2
Creole Mustard
Bottom piece of bread
```

A possible merge of `list.txt`

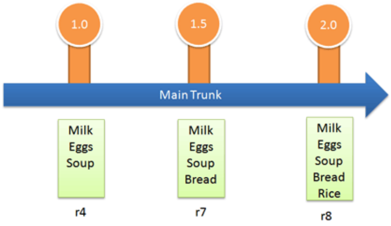
```
Top piece of bread
Mayonnaise
Lettuce
Tomato
Provolone
Salami
Mortadella
Prosciutto
Sauerkraut
Grilled Chicken
Creole Mustard
Bottom piece of bread
```

Finally, removing the 'conflict' tag: `svn resolved list.txt`

Tagging

In Subversion, **tagging** is equivalent to copy the `trunk/` somewhere in `tags/`.

```
module
  |-- branches
  |-- tags
  |    |-- module-1.0
  |    |-- module-1.5
  |    `-- module-2.0
  `-- trunk
```



Releasing your software

1. Warn other developers that the trunk is in '*feature freeze*'.
2. Create a branch '*release candidate*':

```
svn copy -m "Creating release branch 1.1.0" \
    http://svn.repository.net/trunk         \
    http://svn.repository.net/branches/mysoft-1.1.0-rc
```

3. Once release candidate has been stabilized, create the release:

```
svn copy -m "Tagging release 1.1.0"                        \
    http://svn.repository.net/branches/mysoft-1.1.0-rc \
    http://svn.repository.net/tags/mysoft-1.1.0
```

Versioning Policies

Version Numbering

Major version number
Denotes an incompatibility with other major numbers.

Minor version number
Denotes a deep change for developers (API, intern data-structure, . . .).

Release number
Adding features, cleaning code, . . .

Patch level
Bug fixing.

Linux Versioning Policy

Linux 2.6.24.2
(major, minor, release, patch-level)

Code Qualities

Alpha/Experimental
Unstable and missing features.

Beta/Testing
Unstable but all features are there.

Feature Freeze
No more feature addition, focus is on debug.

Release Candidate
Serious candidate for the next release.

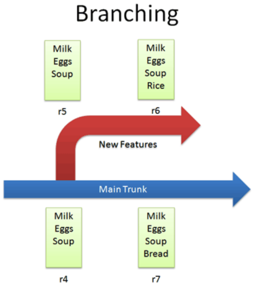
Release/Stable
Code is stable enough to be released.

Branches

Branching

A branch is needed to try some dangerous ideas outside of the main stream development track. So that both can be developed in parallel. Creating a branch is performed by copying the trunk in `branches/` and giving it a name:

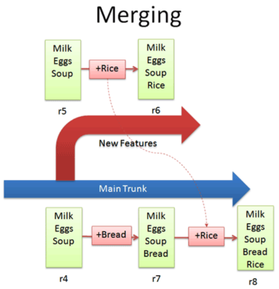
```
svn copy http://path/to/trunk http://path/to/branch
```



Merging

When a branch is stable enough, one wants to merge it back into the main trunk. But main trunk has also evolved, so you need to merge all differences together. Merge is this operation and is performed this way:

```
svn merge -r5:6 http://path/to/branch
```



Advanced Subversion Commands

- `copy`: Copy a file or directory from one point of the main repository to another.
- `export`: Export source code without meta-data.
- `merge`: Attempt to merge two different branches.

Merge Example

Merging a branch of 'project-test' back in the trunk (we are in trunk):

```
svn merge http://svn.depot.net/trunk/project \
          http://svn.depot.net/branches/project-test project
```

- `switch`: Switch from one repository to another with another URI.

Switch Example

Transform a local repository into a branch:

```
svn switch http://svn.depot.net/branches/test ./
```

How to deliver a homework ?

1. **Check that your repository is working**:

```
$> ~efleury/check-svn /path/to/svn/repository
```

2. **Get the revision number of your current version**:

```
$> svn info

 Path: .
 Working Copy Root Path: /home/fleury/MySubversion
 URL: https://svn.labri.fr/fleury/trunk
 Repository Root: https://svn.labri.fr/fleury
 Repository UUID: 83ef70ed-ae93-4ed1-bb18-a6e4ad0ded21
 Revision: 5133
 Node Kind: directory
 Schedule: normal
 Last Changed Author: fleury
 Last Changed Rev: 5133
 Last Changed Date: 2013-09-14 12:28:16 +0200
```

3. **Send me**:
   - The **absolute path** to your repository;
   - The **revision number** of your homework.

How to deliver a homework ?

1. **Check that your repository is working**:

```
$> ~efleury/check-svn /path/to/svn/repository
```

2. **Get the revision number of your current version**:

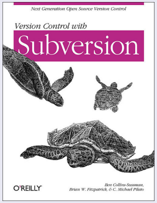
```
$> svn info

 Path: .
 Working Copy Root Path: /home/fleury/MySubversion
 URL: https://svn.labri.fr/fleury/trunk
 Repository Root: https://svn.labri.fr/fleury
 Repository UUID: 83ef70ed-ae93-4ed1-bb18-a6e4ad0ded21
 Revision: 5133
 Node Kind: directory
 Schedule: normal
 Last Changed Author: fleury
 Last Changed Rev: 5133
 Last Changed Date: 2013-09-14 12:28:16 +0200
```

3. **Send me**:
    - The **absolute path** to your repository;
    - The **revision number** of your homework.

Further Readings

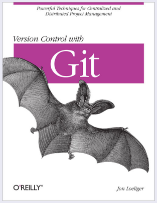
université de BORDEAUX

Version Control with Subversion (2^{nd} Edition)


Next Generation Open Source Version Control
Version Control with
Subversion
O'REILLY
Ben Collins-Sussman, Brian W. Fitzpatrick, & C. Michael Pilato

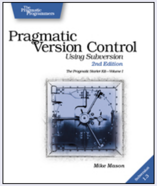
C. Michael Pilato,
Ben Collins-Sussman
Brian W. Fitzpatrick,
O'Reilly, 2008.
(Free On-line book)

Version Control with Git


Powerful Techniques for Centralized and Distributed Project Management
Version Control with
Git
O'REILLY
Jon Loeliger

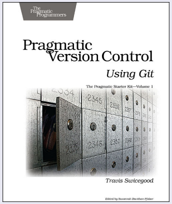
Jon Loeliger,
O'Reilly, 2009.

Pragmatic Version Control using Subversion (2^{nd} Edition)


The Pragmatic Programmers
Pragmatic Version Control
Using Subversion
2nd Edition
The Pragmatic Starter Kit—Volume I
Mike Mason

Mike Mason,
Pragmatic Bookshelf,
2006.

Pragmatic Version Control using Git


The Pragmatic Programmers
Pragmatic Version Control
Using Git
The Pragmatic Starter Kit—Volume 1
Travis Swicegood

Travis Swicegood,
Pragmatic Bookshelf,
2008.

Images on slides 14, 15, 18, 20, 22 are from:
`http://betterexplained.com/articles/a-visual-guide-to-version-control/`