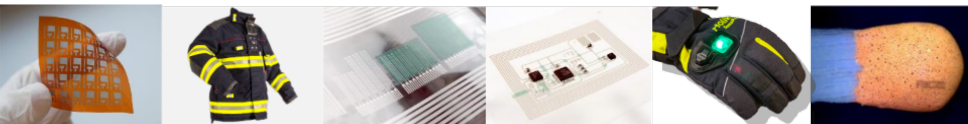


KET/KTL 2020

Introduction to Versioning and Standard Tools for Programmers
Overview with Examples

Jan Bělohoubek

FACULTY OF ELECTRICAL ENGINEERING
UNIVERSITY OF WEST BOHEMIA

FACULTY OF ELECTRICAL ENGINEERING
UNIVERSITY OF WEST BOHEMIA

1 Aims of this Lecture
- Motivation (Anketa)

2 C Overview/Recall

3 Build Automation

4 Version Control Systems

5 Hands-On

FACULTY OF ELECTRICAL ENGINEERING
UNIVERSITY OF WEST BOHEMIA

- provide an **overview** on "classical" engineering/programmer SW tools, namely VCS
- no more "What is this?" !
- hands on . . . but gently
  - our time is limited
  - speed-up the learning curve as you will face the tool

Area/Tool	GNU/Linux	CLI	Scripting	VCS	Make	SSH
Embedded SW	maybe	probably	of course	of course	of course	maybe
Application SW	maybe	probably	of course	of course	of course	probably
Digital Design	probably	probably	of course	of course	probably	probably
PCB Design	maybe	maybe	probably	of course	maybe	maybe
Mechanical Design	maybe	maybe	maybe	probably	rather not	rather not
Research+[1]	of course	of course	of course	of course	of course	of course

[1] Simulations, Super-computing, Networking, Control, AI . . .

1 Aims of this Lecture

2 C Overview/Recall
- About C . . .
- C Compilers
- Compiler Directives
- C Compilation

3 Build Automation

4 Version Control Systems

5 Hands-On

- C is imperative procedural language
  - describes algorithms and uses functions (procedures)
  - low-level system programming language (created for UNIX development)
- C has simple design (close to assembly language)
  - weak types, pointers & pointer arithmetic, pre-processor, inline assembler, . . .
  - Basic types: int, float, enum
  - Derived types: array, pointer, struct, union
- C is everywhere (small & easy-to-develop compilers)
  - C is portable (but platform dependent)
  - C compiler is today the must for any platform (from MCU to supercomputer)

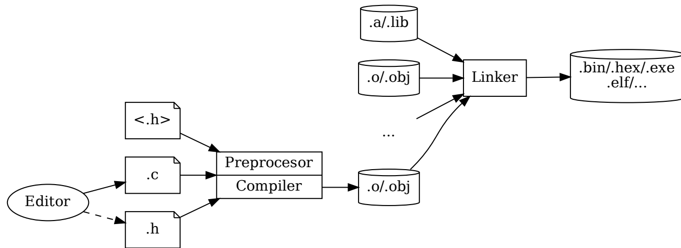
- **GNU C Compiler** – based on (root of the) GNU Compiler Collection
- Clang – based on LLVM compiler infrastructure (originally *Low Level Virtual Machine*)
- ARM Compiler – armcc by Keil (today part of ARM)
- IAR C/C++ Compilers
- Intel C/C++ compiler – Intel targets only (IA-32, IA-64, x86-32, x86-64, Intel Xeon Phi coprocessor, ...)
- ...

- compilers supports number of standards and especially, they have dialects . . .
- sometimes it is required to support more compilers or move from one compiler to another
- it is possible to detect compiler/version in code – **Predefined macros**
- **Pragma** – provide additional information to compiler (C std.)
- **Attributes** – special properties of functions, variables, struct members, . . . (non standard)

```
# dialects and standards ...
$ man gcc
...
$ man clang
...
```

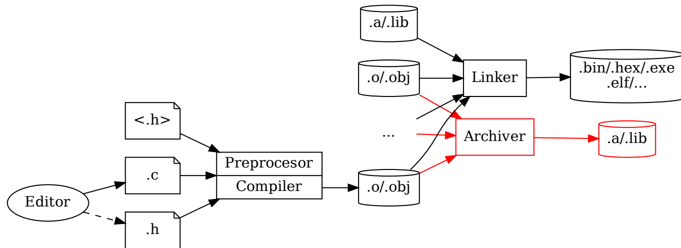
```c
#ifdef __GNUC__
// do not produce unused-variable warning in GCC
__attribute__((unused))
#else
// Terminate compilation for unsupported compiler
#error Unsupported compiler!
#endif

// enable anon unions in Keil
#ifdef __ARMCC_VERSION
#pragma anon_unions
#endif
typedef struct nvm_cfg {
    union {
        struct {
            uint8_t severity;  /*!< msg severity */
            uint8_t calib;     /*!< calib coef. */
        };
        uint8_t RFU[8];        /*!< padding: 8 bytes reserved; 6 b
    };
} nvm_cfg_t;
```

FACULTY OF ELECTRICAL ENGINEERING
UNIVERSITY OF WEST BOHEMIA



- complex process
- files have dependencies

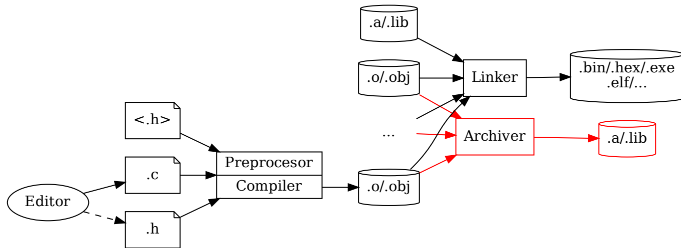
→ bigger projects require automation



- complex process
- files have dependencies

→ bigger projects require automation

FACULTY OF ELECTRICAL ENGINEERING
UNIVERSITY OF WEST BOHEMIA



- complex process
- files have dependencies

→ bigger projects require automation

FACULTY OF ELECTRICAL ENGINEERING
UNIVERSITY OF WEST BOHEMIA

1 Aims of this Lecture

2 C Overview/Recall

3 Build Automation
- Motivation and Brief History
- The Gentle Introduction to GNU/Make

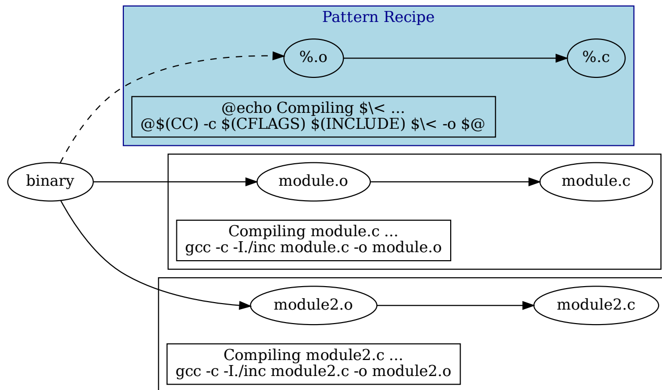
4 Version Control Systems

5 Hands-On

- every larger (software) project needs to define a build process – "Build"/"Rebuild All" buttons in your IDE
- project files describe how to translate source code
- Makefile is the project file interpreted by the Make program
- Unix Make (1976) – GNU/Make, BSD Make, MS nmake
- why to use (GNU/)Make today:
  - de-facto standard compared to ad-hoc build scripts
  - very standard tool – compatible with (or incorporated in) many tools, IDEs, cMake ...
  - human-readable
  - customizable – compared to many native project files – create multiple targets, ...
- other tools: Ninja (2012)

- tool which controls the generation of non-source files from the source files
- make compares target/dependencies timestamps!
- has built-in functions
- uses TAB as a delimiter (!!!)
- Macros:
  - $@ – refers to the target
  - $< – refers to the first dependency
  - $^ – refers to all dependencies
  - % – make a pattern that we want to watch in both the target and the dependency

```
# dependencies and targets are files by default
target: dependencies ...
	system commands
	...
```


Pattern Recipe
%.o
%.c
@echo Compiling $\< ...
@$(CC) -c $(CFLAGS) $(INCLUDE) $\< -o $@
binary
module.o
module.c
Compiling module.c ...
gcc -c -I./inc module.c -o module.o
module2.o
module2.c
Compiling module2.c ...
gcc -c -I./inc module2.c -o module2.o

```
TARGET=envi
BUILD=./build/

OPTIMIZE = -Os
LDFLAGS  =
CFLAGS   = $(OPTIMIZE) -DMODULE_RH -DMODULE_TEMP

CC=gcc
LD=gcc

INC_DIRS = .
INCLUDE  = $(addprefix -I, $(INC_DIRS))

SRCS  = main.c temp.c rh.c
OBJS  = $(addprefix $(BUILD),$(notdir $(SRCS:.c=.o)))

# named recipes - unconditioned execution
.PHONY: all clean

# pattern-matching-based recipe
$(BUILD)%.o: %.c
	@echo Compiling $<                     # @: do not echo command, only execute
	$(CC) -c $(CFLAGS) $(INCLUDE) $< -o $@ # echo command, then execute

all: $(OBJS)
	$(LD) $(OBJS) $(LDFLAGS) -o $(TARGET)

clean:
	-rm -f $(BUILD)*.o $(TARGET)          # -: ignore error
```

FACULTY OF ELECTRICAL ENGINEERING
UNIVERSITY OF WEST BOHEMIA

1 Aims of this Lecture

2 C Overview/Recall

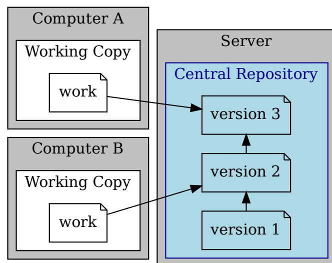
3 Build Automation

4 Version Control Systems
- Motivation . . .
- Brief History . . . Milestones (Open Systems)
- Centralized VCS - CVS, SVN
- Distributed VCS – GIT, Mercurial
- VCS Model and Terminology
- (Our) GIT Reference

5 Hands-On

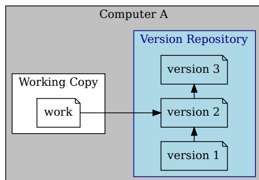
- keep past file(s) versions
- save memory – keep only differences (patches)
- systematic and automated method for versioning (hand-managed structures are error-prone)
- today systems allows to:
  - browse, restore past file versions, record project history
  - conflict resolution
  - colaboration tools
  - . . .
- GitHub, Bitbucket, GitLab, SourceForge, . . . :
  - additional tools - forum, issue tracker, wiki, . . .
  - central server
  - . . .

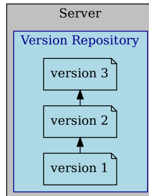
- local versioning (70s +)
  - Source Code Control System (SCCS) – 1973
  - Revision Control System (RCS) – 1972
- centralized versioning (90s +)
  - centralized model allows synchronization and centralized administration
  - Concurrent Versions System (CVS) – RCS extension; 1990
  - Subversion (SVN) – designed to replace CVS; 2000
- distributed versioning (00s +)
  - distributed model removes single-point-of-failure and enables different workflows
  - Git – created for Linux Kernel; 2005
  - Mercurial – 2005

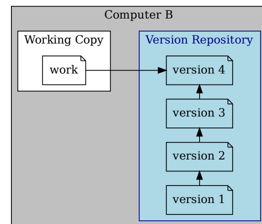


Computer A
Working Copy
work
Computer B
Working Copy
work
Server
Central Repository
version 3
version 2
version 1

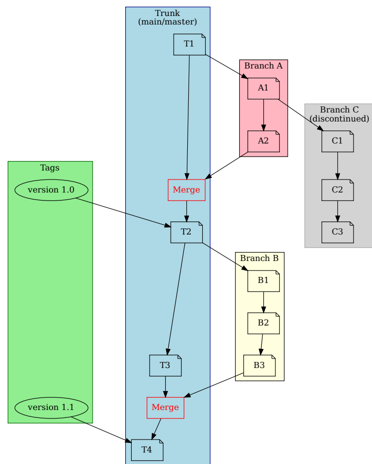
Version Control Systems
Distributed VCS – GIT, Mercurial







Version Control Systems
VCS Model and Terminology



https://en.wikipedia.org/wiki/File:Revision_controlled_project_visualization-2010-24-02.svg

FACULTY OF ELECTRICAL ENGINEERING UNIVERSITY OF WEST BOHEMIA

- Create repository: git init [PATH]
- Clone repository: git clone URL
- Select file for commit (add to staging area): git add FILEs
- Show branches: git branch
- Get working tree status: git status
- Create a new commit[2]: git commit
- Browse history: git log
- Switch branches: git checkout
- Update remote repository: git push
- Get changes from remote repository: git pull
- Join branches: git merge BRANCH

[2] Commit – submit the latest changes to the repository

FACULTY OF ELECTRICAL ENGINEERING
UNIVERSITY OF WEST BOHEMIA

1 Aims of this Lecture

2 C Overview/Recall

3 Build Automation

4 Version Control Systems

5 Hands-On
- Get Remote and Discover Makefile
- Modify Files
- Resolve Conflicts

Hands-On
Get Remote and Discover Makefile


```
$ git clone https://github.com/belohoub/KTL.git
Cloning into 'KTL'...
remote: Enumerating objects: 17, done.
remote: Counting objects: 100% (17/17), done.
remote: Compressing objects: 100% (12/12), done.
Receiving objects: 100% (17/17), done.
remote: Total 17 (delta 3), reused 17 (delta 3), pack-reused 0
Resolving deltas: 100% (3/3), done.
$
$ cd KTL && ls -l # go to your working copy and list files (.git)
$
$ git branch
* master
$
$ git checkout multipleTargets
Switched to branch 'multipleTargets'
$
$ git branch
  master
* multipleTargets
```

Hands-On
Get Remote and Discover Makefile


```
$ # Display all commits
$ git log
$ # display "patch" (difference) between last two commits
$ git log -p -1
$ # display log in short
$ git log --pretty=oneline --abbrev-commit
$
$ # display graph - see later
$ git log --pretty=format:"%h %s" --graph
...
$
$ # create new branch from 'master'
$ git checkout master
Switched to branch 'master'
Your branch is up to date with 'origin/master'.
$ git checkout -b myFirstBranch
Switched to a new branch 'myFirstBranch'
$ git branch
  master
  multipleTargets
* myFirstBranch
```

FACULTY OF ELECTRICAL ENGINEERING
UNIVERSITY OF WEST BOHEMIA

```
$ # Show branch status
$ git status
On branch myFirstBranch
nothing to commit, working tree clean
$
$ # create new file and track it
$ echo "#define␣TEMPERATURE␣45" > src/defines.h
$ git status
On branch myFirstBranch
Untracked files:
  (use "git␣add␣<file>..." to include in what will be committed)
        src/defines.h
...
$ git add src/defines.h
$ git status
...
$ # add include to temp.h
$ echo "#include␣\"defines.h\"" | cat - src/temp.c > \
    src/temp.c~ && mv src/temp.c~ src/temp.c
$
```

```
$ # modify printf
$ awk '{if␣(NR␣==␣10){␣print␣"printf(␣\
␣␣␣␣\"␣␣-␣temperature␣is␣%d\\n\",␣TEMPERATURE);"}␣\
␣␣␣␣else{print␣$0}}' src/temp.c > src/temp.c~ && \
    mv src/temp.c~ src/temp.c
$
$ git status
On branch myFirstBranch
Changes to be committed:
        new file:   src/defines.h
Changes not staged for commit:
        modified:   src/temp.c
$
$ # add src/temp.c
$ git add src/temp.c
$
$ git commit -m "true␣temperature"
[myFirstBranch 5fb2ede] true temperature
 Date: Sun Nov 22 08:08:02 2020 +0100
 2 files changed, 3 insertions(+), 1 deletion(-)
 create mode 100644 src/defines.h
```

Hands-On
Resolve Conflicts


```
$ # display "patch" (difference) between last two commits
$ git log -p -1
...
$
$ # compare two branches
$ git diff myFirstBranch..multipleTargets
$
$ # merghe branches
$ git merge multipleTargets
Auto-merging src/temp.c
CONFLICT (content): Merge conflict in src/temp.c
Automatic merge failed; fix conflicts and then commit the result.
$
$ git status
On branch myFirstBranch
You have unmerged paths.
Changes to be committed:
        modified:   Makefile
Unmerged paths:
        both modified:   src/temp.c
$
```

```
$ # resolve conflicts manually
$ vi src/temp.c
...
$
$ # try our new software
$ make both
$ ./envi_temp
This is main() BEGIN
Enviromental parameters:
  - temperature is 45
^C
$
```

```
$ add changes & commit
$ git add src/temp.c
$ git commit -m "Merging two branches"
[myFirstBranch 928b928] Merging two branches
$
$ # enjoy graph :-)
$ git log --pretty=format:"%h %s" --graph
*   928b928 Merging two branches
|\
| * 77f0285 Customer-specific targets were added; ...
* | 5fb2ede true temperature
|/
* 2aa0cee Init
$
$
$ # try to push
$ git push
...
$ git push --set-upstream origin myFirstBranch
 ... [ you do not have permission to push ]
$
```

- Featured Reading and Resources:
  - https://archive.org/details/TheCProgrammingLanguageFirstEdition/mode/2up
  - Herout, Pavel. Učebnice jazyka C, 3. vyd. Kopp: 1994
  - https://gcc.gnu.org/onlinedocs/gcc/
  - https://www.gnu.org/software/make/
  - https://git-scm.com/book/en/v2
  - https://guides.github.com/introduction/git-handbook/
  - https://learnxinyminutes.com/docs/git/

Thank you for your attention!

Jan Bělohoubek
UWB, Czech Republic
belohoub@fel.zcu.cz
+420 377 63 4514

FACULTY OF ELECTRICAL ENGINEERING
UNIVERSITY OF WEST BOHEMIA