

Software Development Tools

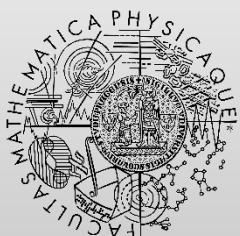
<http://d3s.mff.cuni.cz>

Department of
Distributed and
Dependable
Systems



Pavel Parízek

parizek@d3s.mff.cuni.cz



FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

Goal of this course



- Basic overview of available tools for common tasks in software development
- Practical experience with selected tools
- This can be useful in
 - Studies: assignments, individual projects, team SW projects, bachelor thesis, master thesis
 - Commercial software development: productivity
 - Work on open-source projects (recommended!!)

Expectations



- User knowledge of UNIX/Linux and Windows
 - Command line (shell), writing small scripts, system utilities, common user applications
 - Practical experience is an advantage (for UNIX/Linux)
- Basic knowledge of a mainstream programming language (C/C++, Java, C#)
 - Extent of introductory course at MFF
 - Sufficient to attend in the same term
 - Practical experience is an advantage

Content



- Version control systems (SVN, Git, Hg) S1
- Software building (Make, Ant, MS Build) S2
- Functional testing (JUnit, NUnit) S3
- Debugging (GDB, Valgrind) S3
- Searching for bugs (FindBugs, Clang, FxCop) S3
- Recording events (strace, log4j) S4
- Issue trackers (Bugzilla, Trac)
- Generating documentation (Doxygen)
- Generating code from templates
- Performance analysis (GProf, JVisualVM) S4

Structure of each lab



- Introduction to a given domain
 - Basic concepts, what problems the tools address
 - Description of selected tools (technical details)
- Specific features of tools
 - Syntax of commands, basic configuration
- Practical tasks
 - Individual work during labs, online documentation
- Homework assignment

Report



- Purpose: show that you
 - Understand the assignment (specific task)
 - Found out how to solve the given task, and
 - Managed to do it successfully in practice
- Form
 - Text file, ASCII, Czech/Slovak or English
 - Attachments (source code, textual outputs)

Report – content



- Your full name and email address
- Specification of individual tasks
 - Copy from the original assignment
 - Put it just before description of your solution
 - The recommended way: edit the file with assignment directly
- Your solution
 - What commands you run (including arguments)
 - Where you run the commands (in which directory, etc)
 - Output of the tool (just important parts): console, files
 - Brief explanation (comment)
 - If there are multiple possible solutions or you decided to use a non-trivial approach

Report – example



John Doe, john@doe.com

1. Create a directory named „test“ in /tmp and discuss situations in which your solution would fail

```
> cd /tmp  
/tmp> mkdir test
```

Creating a directory in this way could fail for these reasons: the directory /tmp does not exist or the current user does not have the effective access privilege “x” on the directory, the current user does not have the privilege “w” on the directory “/tmp”, a physical device mapped to the directory /tmp is full (i.e., there is not enough space for a new directory).

2. ...
-

3. ...

Grading



- Reports
 - Important aspects: clarity, correctness, participation at the lab
 - Unsatisfactory reports → one more week for resubmission
- Credit
 - Regular attendance (75%)
 - You can do additional homework assignments (over the minimum of 6) to compensate for skipped labs
 - Homework assignments (6 out of 9)
 - At least one assignment from each group of topics (S1-S4)
 - Attended the lab: basic variant of a homework assignment
 - Missed the lab: few additional tasks (bit more advanced)
- Alternative means of fulfilling the course
 - Skipped labs → more complex tasks over real systems
 - Usage of a tool on a student project (describe your experience)

My vision for the whole course



- Labs: focus on activity of students (practical tasks)
- Interactive mode of teaching: questions, higher student activity
 - Much less frontal lecturing (teacher standing before the class)
- Students should work (play with the tools, get experience)
 - „Controlled self-study where I can help to a large extent“
- **Do not be afraid to ask (!!)**, when something does not work for you
- I speak quite fast so be patient with me (but I am trying to slow down)
- Attendance is **optional**: when you skip some lab, you may get an additional tasks for homework to compensate
- Three groups of tools
 - Very common: you should certainly know all of them and be able to use them
 - Interesting: also important tools that you should be aware of (in my opinion)
 - Other tools: general overview (so you know what to look for in time of need)
- Sometimes we do not manage to go through the whole presentation in the lab => please try the rest at home
- Sample solutions discussed very rarely (almost never)
- Tweaking the content and form every year based on your feedback

Contact



- Web: <https://d3s.mff.cuni.cz/teaching/nswi154/>
- Email: parizek@d3s.mff.cuni.cz
- Office 202

Version Control

(Správa verzí)

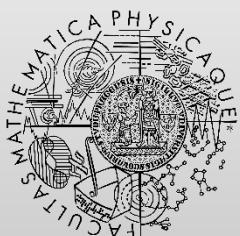
<http://d3s.mff.cuni.cz>

Department of
Distributed and
Dependable
Systems



Pavel Parízek

parizek@d3s.mff.cuni.cz



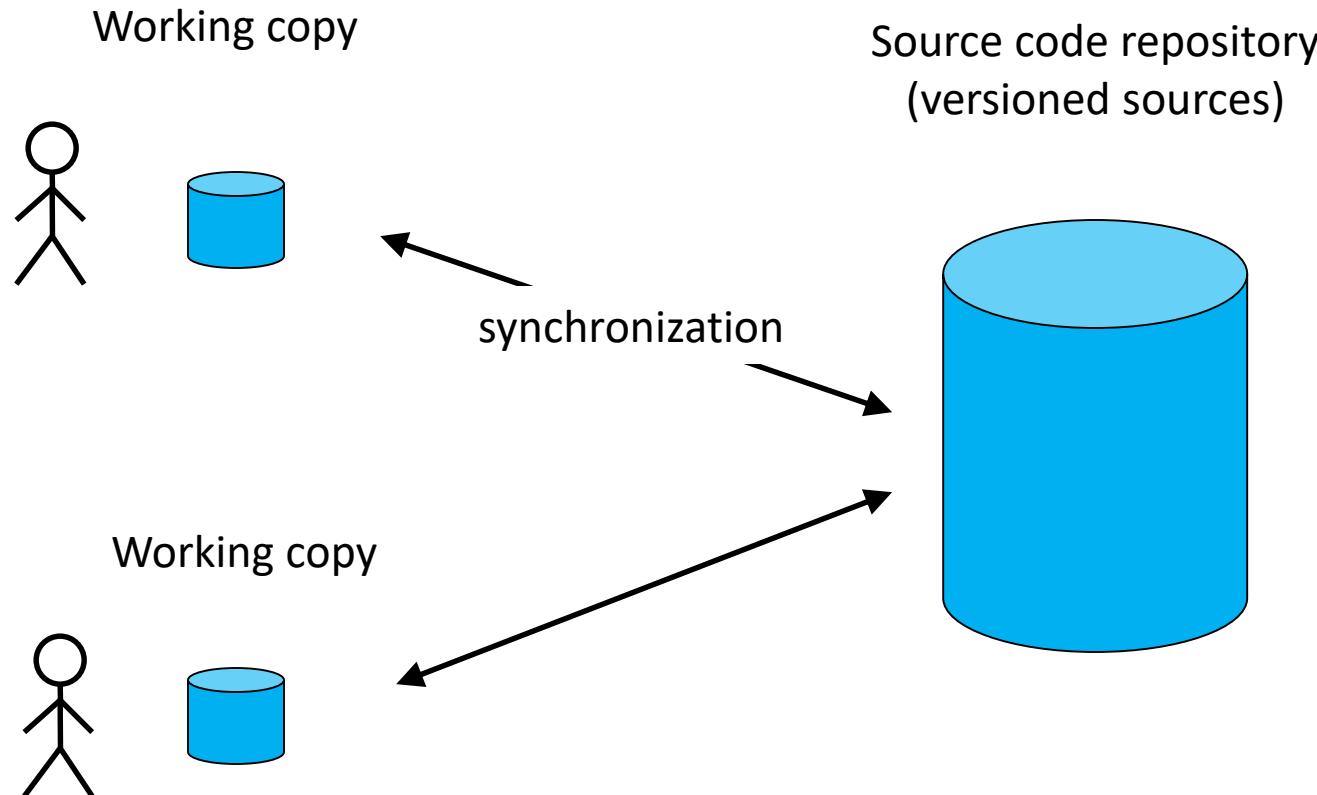
FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

What is it good for ?

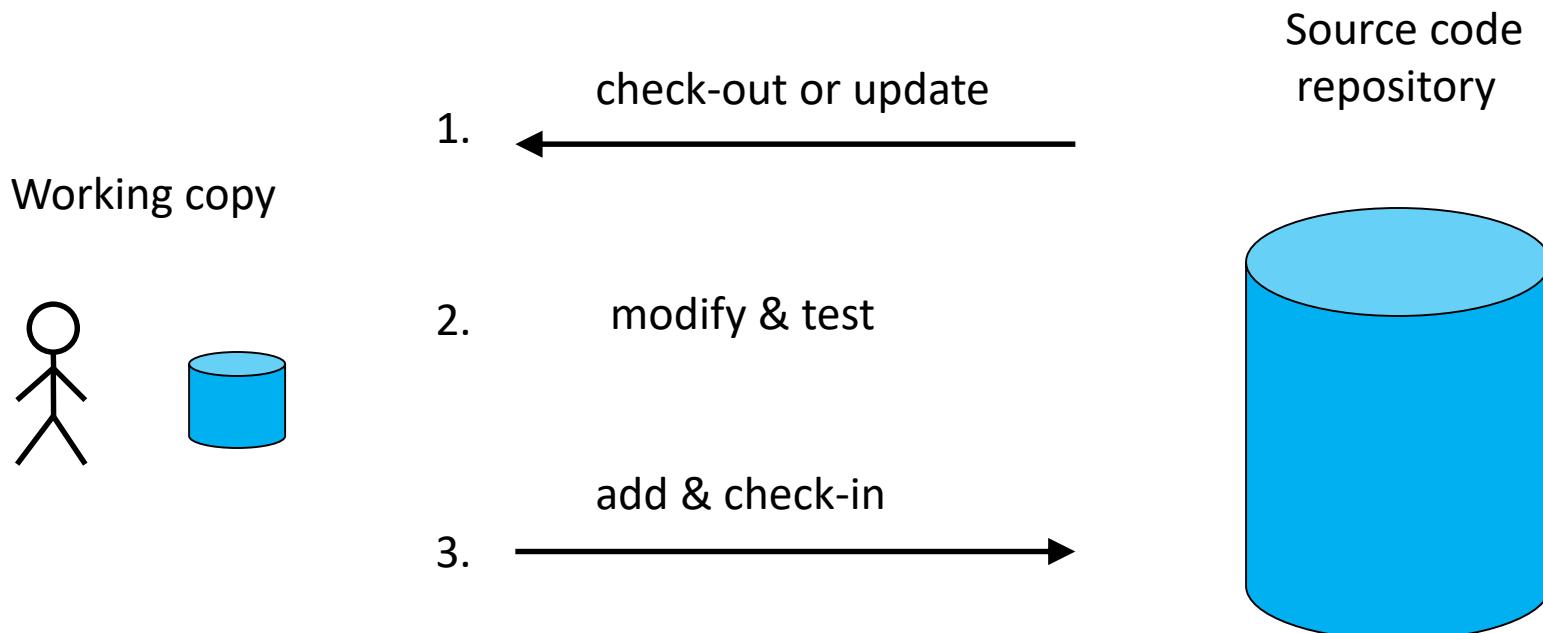


- Keeping history of system evolution
 - Tracking progress
- Allowing concurrent work on the system
 - Teams of developers
 - Possible conflicts
- Easy reverting to a previous version
 - Safer experimentation

Typical architecture



Basic usage scenario

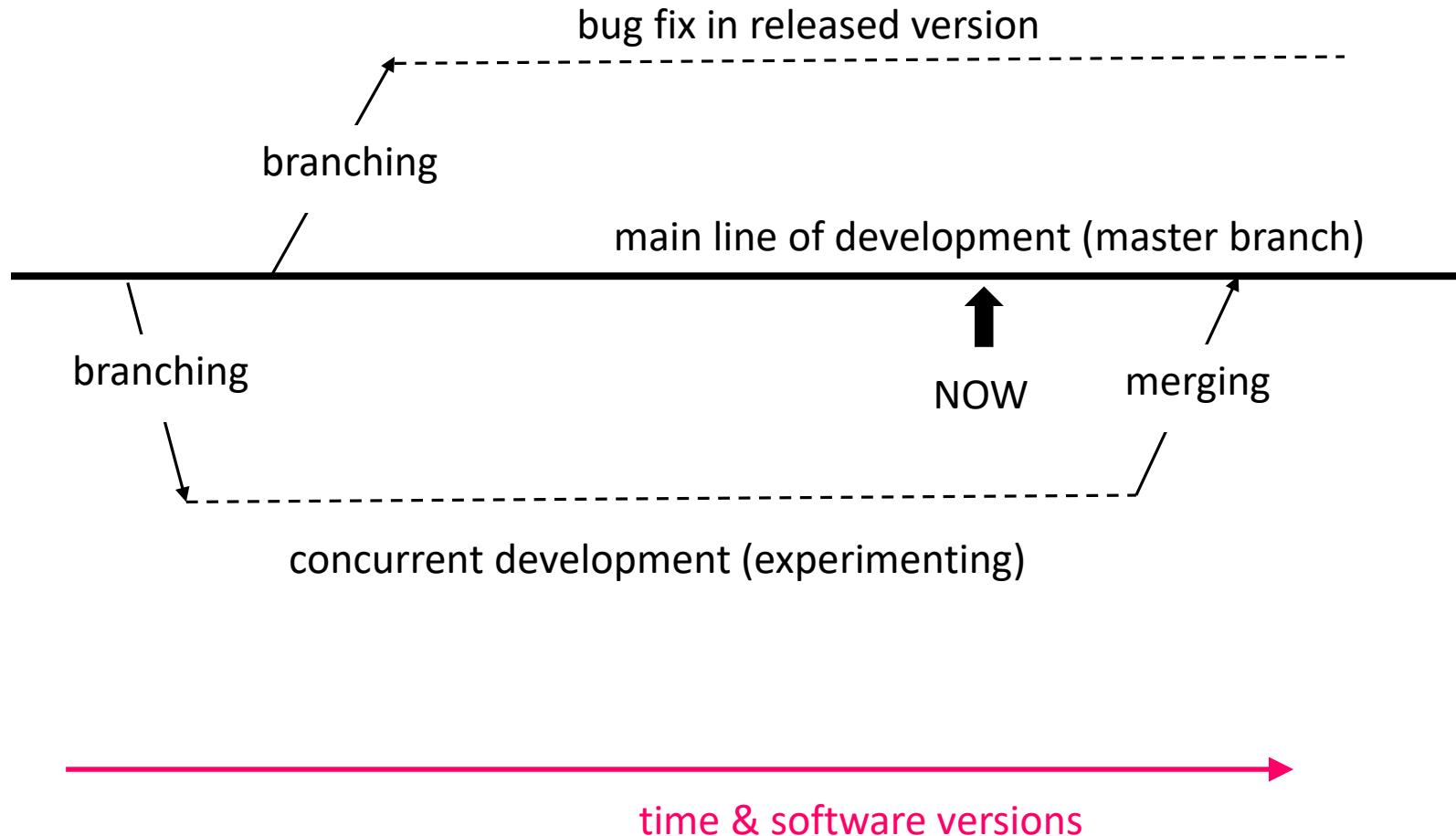


Categories of versioning systems



- Centralized
 - CVS: Concurrent Versioning System
 - The “classic” system
 - SVN: Subversion
 - Currently still used by many open-source projects
 - <http://apache.org/index.html#projects-list>
- Distributed
 - Git, Mercurial, Bazaar

Branches and merging



Conflicts



- Options
 - Postpone resolving
 - Choose version
 - External merge tool
 - and many others
- Conflict markers
 - <<<<<< and >>>>>> in source file
- Three variants of the source file created

Tree conflicts



- Typical cause
 - Renamed files and directories
 - Deleted files
- Solution
 - Make proper changes in the working copy
 - Use patches created with the `diff` command
 - Commit when everything is in a clean state

Tags



- Snapshot with a human-friendly name
- Logical copy of the whole source tree

Best practices: synchronizing developers



- Software developed in large teams
 - People may not be always able to coordinate efficiently
- Solution: Copy-Modify-Merge
 - Concurrent modification of source files
 - Resolving conflicts when they happen
- Alternative: Lock-Modify-Unlock
 - The old classic approach (“before internet”)
 - Does not scale well (exclusive access)
 - Not very robust (people forget to unlock)

Best practices: branches and merging



- Use branches for experimental features
- Create special branch for each feature
- Separate release and development branches
 - Propagating bugfixes from development to stable
- Merge often and synchronize with trunk
 - Lower chance of ugly conflicts occurring
 - Smaller conflicts are easier to resolve
 - Commit often → others will have to merge

Subversion



Important features



- Whole source tree versioned
 - Integer numbers (1,2,3,...)
- Mixed versions in the working copy
- Atomic commits
- Versioning for files and directories
 - Operations: move, rename, delete
- Support for binary files
- Disconnected operations
- Metadata in “.svn” directories

Locations



- Repository
 - Remote server (`svn+ssh://<network url>`)
 - Local directory (`file://<absolute path>`)
- Working copy
 - Local directory on your computer

Basic commands



- Help: `svn help <command>`
- Create new repository: `svnadmin create`
- Create new working copy: `svn checkout`
- Update working copy: `svn update`
- List modified and new files: `svn status -v`
- Show differences between repository and working copy (two versions): `svn diff -r<version>`
 - Use `svn diff -r<v1>:<v2>` to see differences between two specific versions
- Add new files into repository: `svn add`
- Commit changes: `svn commit -m "..."`
- Display information about file: `svn info`

Few more useful commands



- Undo changes in working copy: `svn revert`
- See full history of a given file: `svn log`
- Importing whole unversioned tree into repository: `svn import <dir> <repo>`
- Exporting content of the repository without metadata: `svn export`

Managing files and directories



- Commands

- svn add <path>
- svn delete <path>
- svn copy <path1> <path2>
- svn move <path1> <path2>
- svn mkdir <path>

- Path

- In your local working copy
- Repository (auto-commit)

Branching and merging – commands



- Create new branch
 - `svn copy <main line repo path> <branch repo path>`
- Print differences
 - `svn diff <main line repo path> <branch repo path>`
- Make your branch up-to-date (sync merge)
 - `svn merge <main line repo path>`
 - `svn merge ^/<main line repo dir>`
- Merge branch into the main line (trunk)
 - `svn merge --reintegrate ^/<branch repo dir>`
- Preview
 - `svn merge <repo path> --dry-run`

Undoing committed modifications



- Merge negative version range into local working copy
 - `svn merge <repo path> -r <v1>:<v2>`
 - Note: $v1 > v2$
- Commit everything

Cherrypicking



- Merge specific change into your branch
 - `svn merge -c <version> <repo path>`
- Commit your branch

Standard repository layout



```
/trunk  
/branches  
/tags
```

```
/project1/trunk  
/project1/branches/feature1  
/project1/tags  
/project2/trunk  
/project2/branches  
/project2/tags/R_1_0  
/project2/tags/R_1_2_1
```

Revision keywords



- HEAD
 - Latest version in the repository
- BASE
 - Revision number in the working copy (before modifications)
- COMMITTED
 - The latest revision in which the item changed and was committed (not larger than BASE)
- PREV
 - Equal to COMMITTED-1

Properties



- Standard name-value pairs
- Many internal system properties
 - `svn:ignore`, `svn:eol-style`, ...
- Setting property value
 - `svn propset <name> <value> <path>`
 - `svn propset <name> -F <file> <path>`
- Other commands
 - `svn proplist`
 - `svn propget`
 - `svn propedit`

Locks



- Still needed to work with binary files
 - Merge not supported for concurrent modifications
- Locking
 - svn lock
- Unlocking
 - svn commit
 - svn unlock
- Inspecting
 - svn info

GUI clients for SVN



- Tortoise SVN (Windows)
 - <http://www.tortoisessvn.net>
- Eclipse IDE
- Other
 - kdesvn (Linux)
 - svnx (Mac OS)

Links

- <http://subversion.apache.org>
- SVN Book
 - <http://svnbook.red-bean.com>

Distributed Version Control

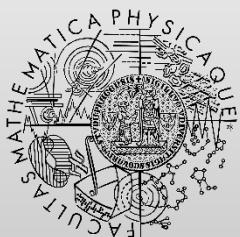
<http://d3s.mff.cuni.cz>

Department of
Distributed and
Dependable
Systems



Pavel Parízek

parizek@d3s.mff.cuni.cz



FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

Key concepts



- Each developer uses a private local repository
 - *clone*: full mirror of some existing repository
- Operations performed on the local repository
 - very fast, off-line
- Synchronization
 - Operations *push* and *pull*
 - Exchanging code patches

Comparing distributed and centralized VCS



- Centralized
 - Everything visible in the central repository
 - Private branches (work) not possible
- Distributed
 - Private repositories (and branches) useful for experimental development

Tools



- Git
- Mercurial
- Bazaar

Git



Main features

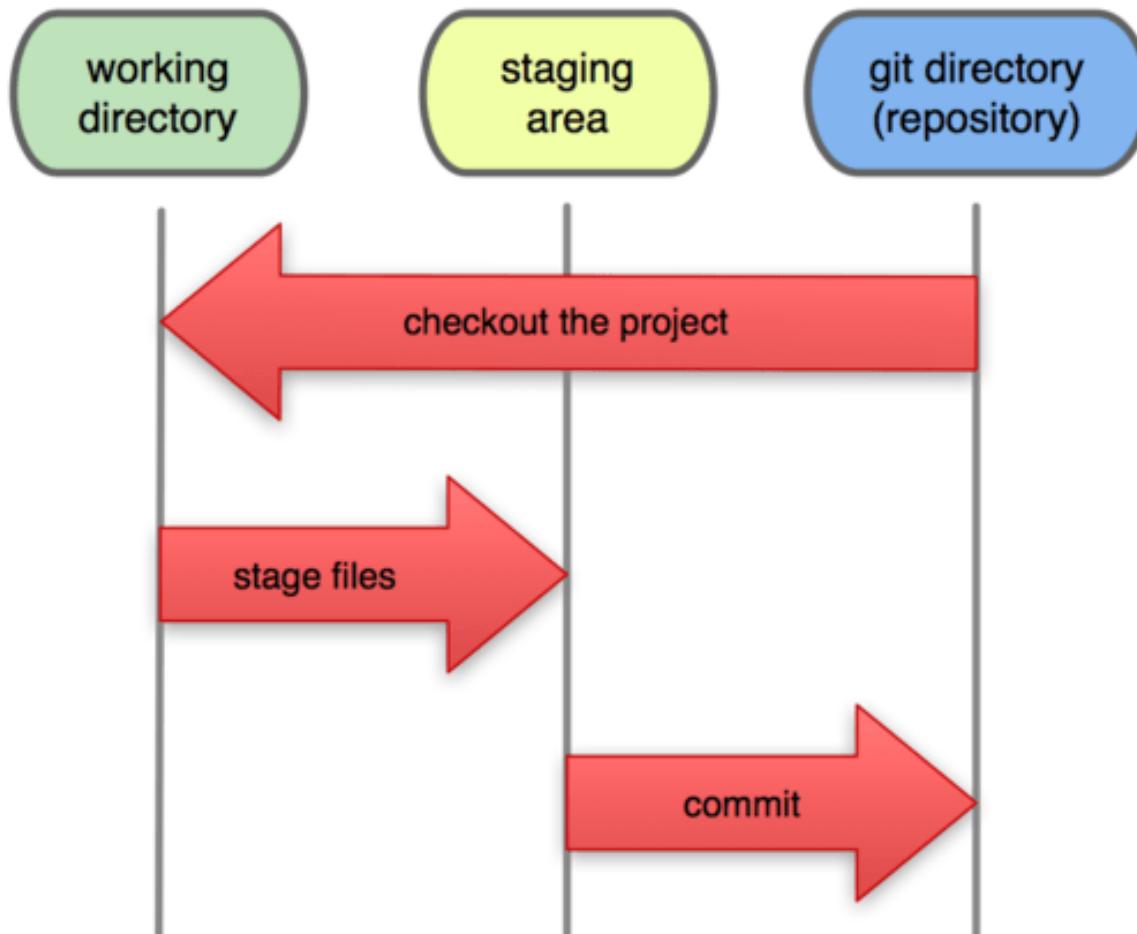


- Versions: snapshots of the project (working dir)
- Committed revisions form a direct acyclic graph
 - Multiple “latest” versions (leaf nodes)
- Each commit has an author and committer
 - Distributing changesets via patches (email)
- Whole repository stored in .git (files, metadata)
- Confusing for most people (good for advanced users)
- Commands have names similar to SVN

Usage scenario



Local Operations



Picture taken from <http://git-scm.com/book/>

Task 1



- Configure your identity
 - `git config --global user.name "your full name"`
 - `git config --global user.email "your email address"`
- Stored in `$HOME/.gitconfig`

Basic commands

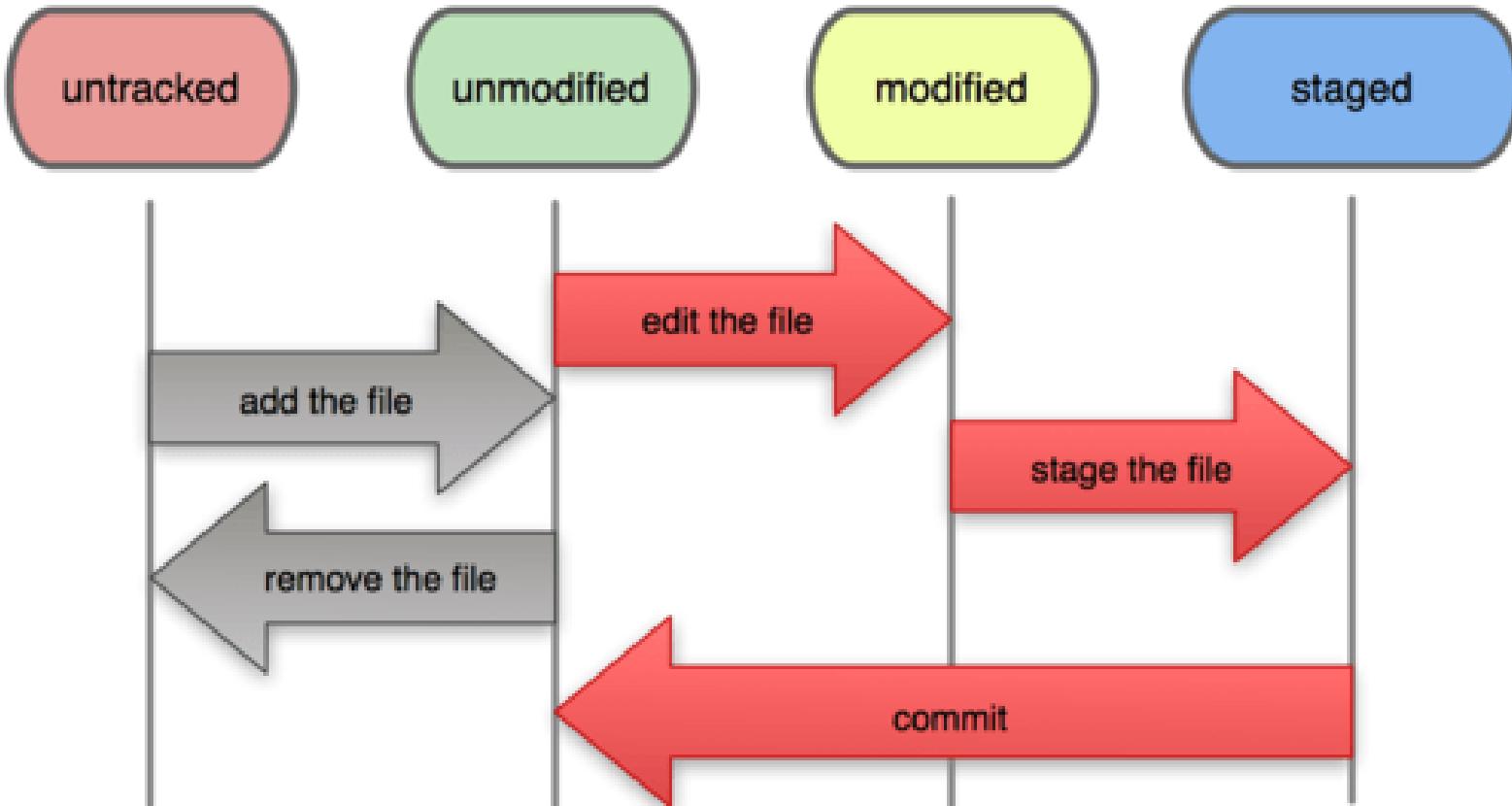


- Create repository in the current directory: `git init`
- Print status of the working tree: `git status`
- Start tracking new files: `git add <work dir path>`
- Add files to the staging area: `git add <path>`
- Commit staged modifications: `git commit -m "..."`
- Print uncommitted unstaged changes: `git diff`
- Print staged uncommitted changes:
`git diff --staged`
- Automatically stage every tracked file and commit
`git commit -a -m "..."`
- Revert modifications: `git checkout -- <path>`

File status lifecycle



File Status Lifecycle



Picture taken from <http://git-scm.com/book/>

Task 2



- Create repository in a specific directory
- Create some new files (e.g., hello world)
- Print current status of your repository and the working directory
- Stage all the new files
- Print current status
- Modify one of the files
- Print current status
 - Inspect differences from the previous invocation
- Commit all staged modifications
- Print current status

Managing files



- Make the given file untracked

```
git rm <work dir path>
```

- Renaming file (directory)

```
git mv <old path> <new path>
```

Pick your changes



- Full interactive mode: `git add -i`
- Select patch hunks: `git add -p`
- Additional information with examples
 - <https://git-scm.com/book/en/v2/Git-Tools-Interactive-Staging>

Project history



- List all the commits

```
git log [-p] [-<N>] [--stat]
```

- More options

```
[--pretty=oneline|short|full|fuller]
```

```
[--graph]
```

```
[--since=YYYY-MM-DD]
```

```
[--until=YYYY-MM-DD]
```

```
[--author=<name>]
```

Task 3



- Try out file management commands (`rm`, `mv`)
- Play with the “`git log`” command
 - Explore different parameters (`-p`, `-<N>`, `--stat`, `--pretty`, `--graph`)
- Run the program “`gitk`” and try it
- Make some changes to a particular file and use interactive staging

Using remote repositories



- Clone a remote repository in the current local directory: `git clone <repo url>`
- Get recent changes in all branches from the remote repository: `git fetch origin`
- Get recent changes in the “master” branch and merge into your working copy: `git pull`
 - Announcements via *pull requests*
- Publish local changes in the remote repository: `git push origin master`

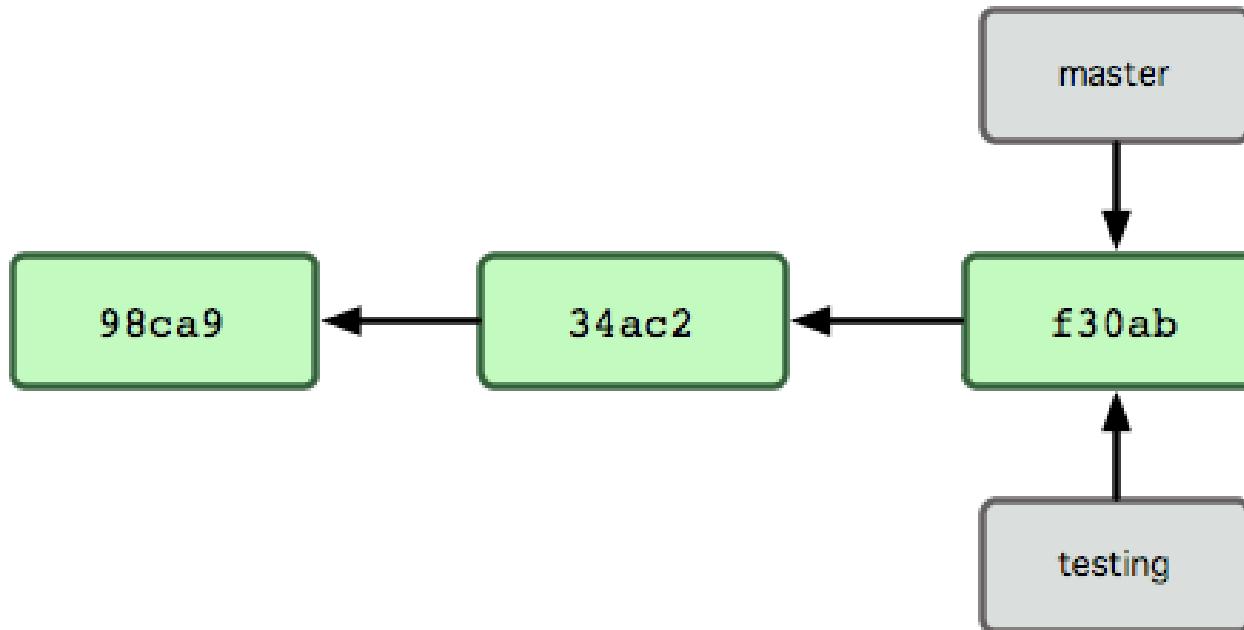
Branches in Git



Branches in Git

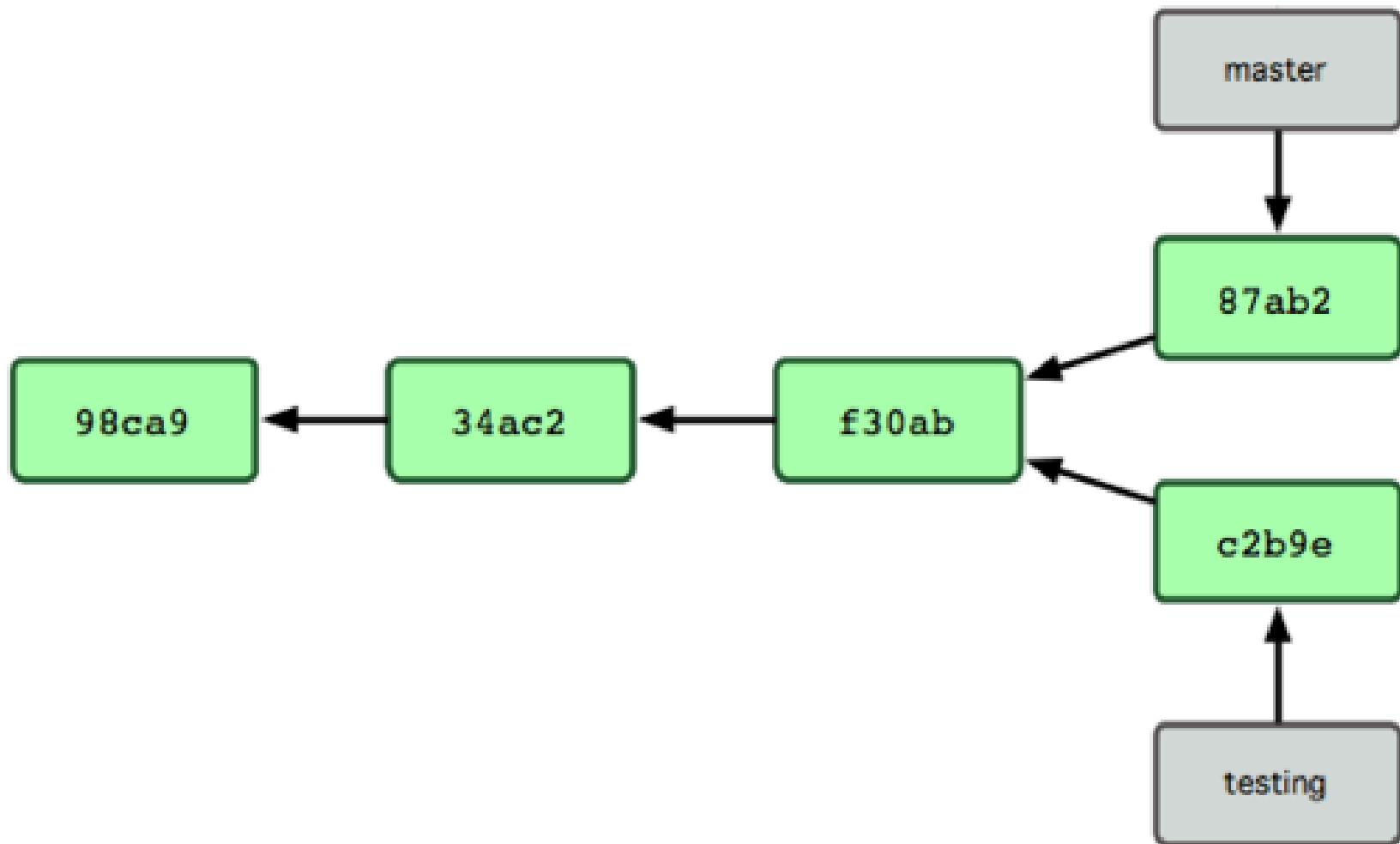


- Branch: pointer to a node in the revision DAG
- Default branch: **master**
- Commit: branch pointer moves forward



Picture taken from <http://git-scm.com/book/>

What happens after concurrent modification



Picture taken from <http://git-scm.com/book/>

Branches in Git: commands



- Create new branch: `git branch <name>`
- Switch to given branch: `git checkout <name>`
- Shortcut: `git checkout -b <name>`
- Merge branch into current working directory
`git merge <branch name>`
- Deleting unnecessary branch
`git branch -d <branch name>`
- List all branches: `git branch [-a]`
 - Current branch marked with *

Comparing branches



- `git diff <branch 1>..<branch 2>`
 - Compare heads of the two branches
 - Note the characters ‘..’
- `git diff <branch 1>...<branch 2>`
 - Print changes on the branch 2 (e.g., master) since the branch 1 (feature) was created from it
 - Note the characters ‘...’

Three-way merge



- Common ancestor
- Target branch
- Source branch
- Conflicts happen also with Git
 - Standard markers <<<<< =====>>>>>
 - Marking resolved files: `git add`
- Graphical merging tool: `git mergetool`

Task 4



- Create new branch B and switch to it
 - Modify some files and commit them
 - Switch back to the master branch
 - Modify some files and then commit
 - Merge your branch B into the master
 - Delete the now unnecessary branch
-
- Try switching branches with uncommitted changes in the working copy
 - Try graphical merging tool on some conflicts

Advanced features I.



- Using stack of unfinished changes (stashing)
 - git stash [push]
 - git stash apply [<stash name>]
 - git stash list
- How to undo some changes
 - git reset <commit>
 - Moves the branch HEAD to a given commit
 - Several variants
 - --soft: undo commit
 - --mixed (default): undo commit and changes in staging area
 - --hard: undo everything (commit, staging area, working dir)

Advanced features II.



- Symbolic names of versions
 - HEAD, HEAD~1, HEAD^2
- git rebase
 - Replaying changes done in a branch onto another branch
 - Very powerful command but also tricky (be careful !!)
- Modifying committed history
 - e.g., commit messages (git commit --amend)
- Ignoring certain files
 - List patterns in the file .gitignore
- Tagging: git tag
- Bare repository
 - No working copy

Mercurial



- Basic principles: like Git
- Simpler learning curve
- Commands very similar
 - init, clone, add, commit, merge, push, pull

Work-flow models (cooperation)

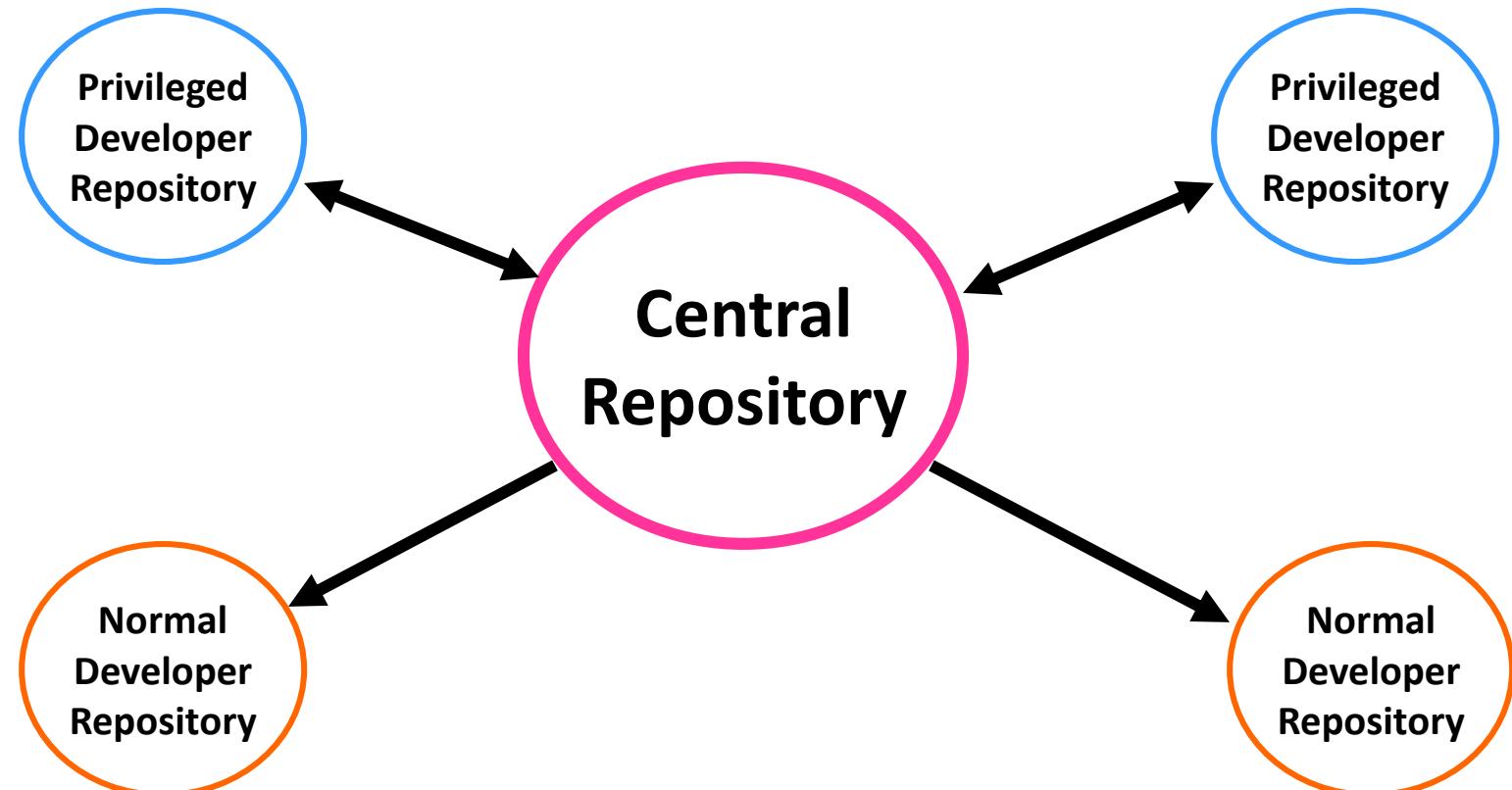


Work-flow models (cooperation)

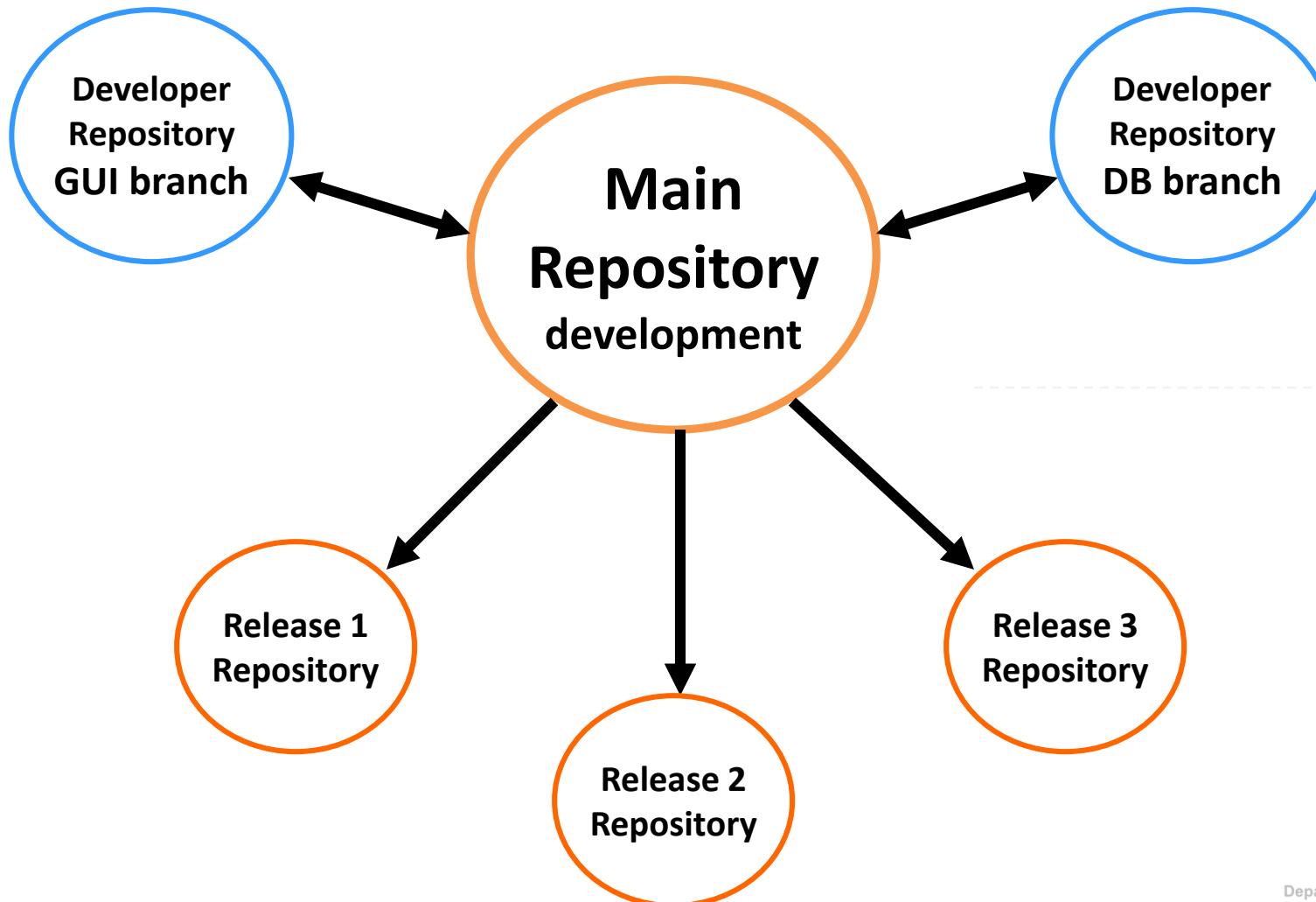


- Anything possible technically with DVCS
- “Network of trust” between developers
- Examples
 - Single “central” repository
 - Multiple release repositories
 - Many public repositories
 - Total anarchy

Single “central” repository



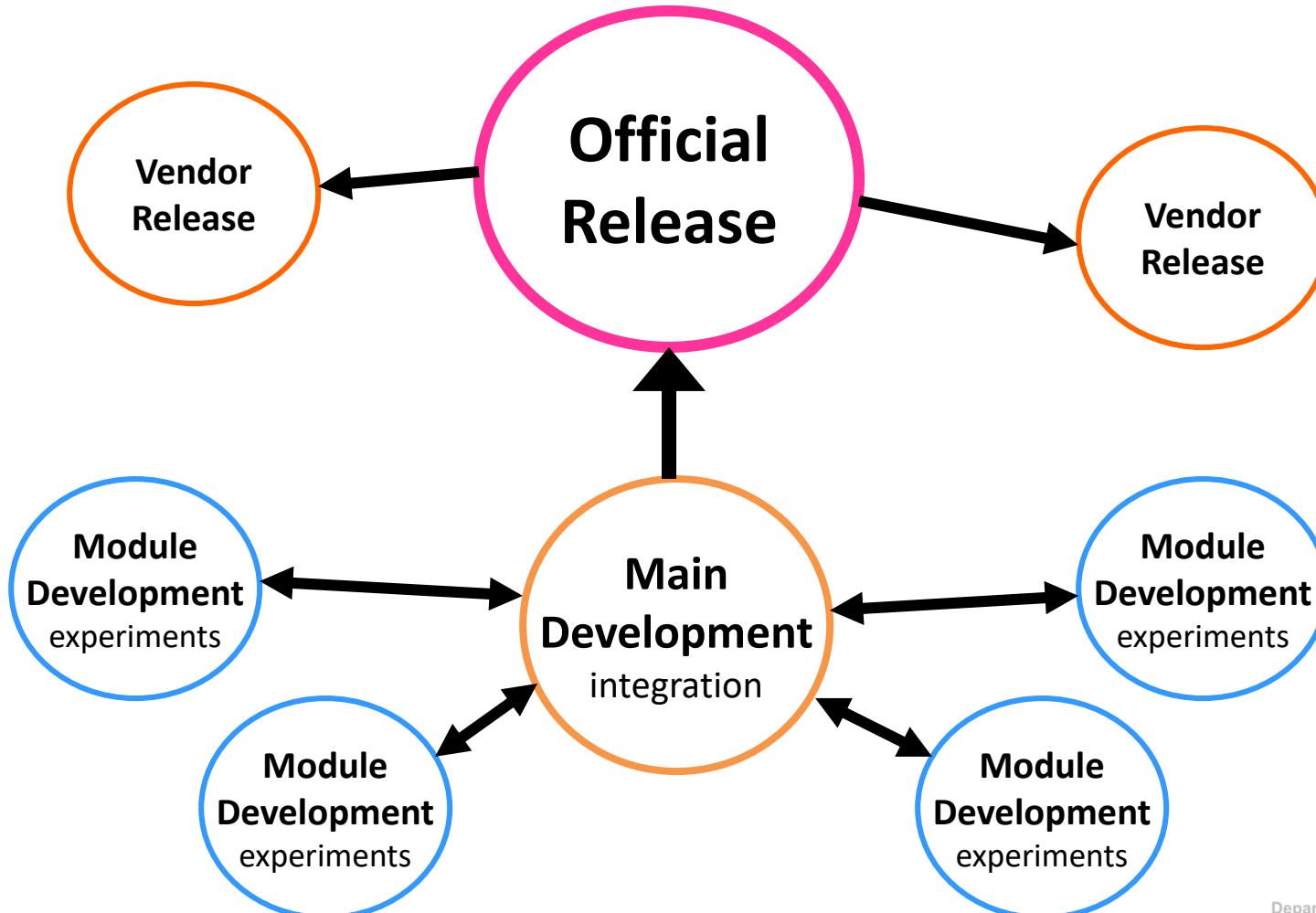
Multiple release repositories



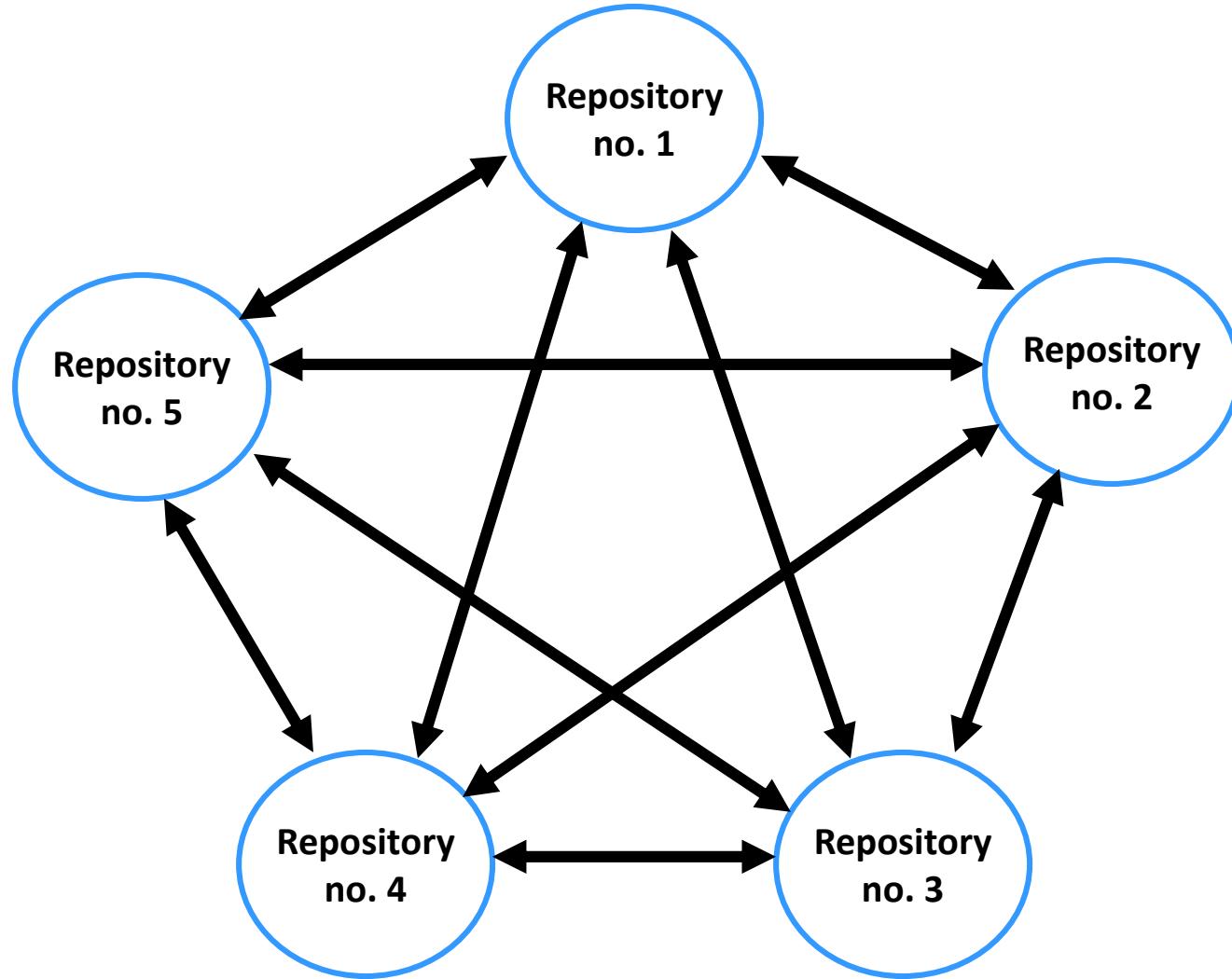
Many public repositories



- Linux kernel



Total anarchy



Contributing to [open-source] projects



- Typical scenario
 - Project hosted on some public repository server
 - Write access to official repository is not possible
- Important concepts
 - Forking of the official repository
 - Publishing via pull requests

Contributing to [open-source] projects



- Official central repository (upstream)
 - <https://github.com/projectname>
- Fork on the same server
 - <https://github.com/user/projectname>
- Clone to local repository
 - From <https://github.com/user/projectname> to
\$HOME/projectname
- Synchronizing fork with official repository
 - git fetch upstream
 - git merge upstream/master
- Publishing changes to the upstream repository
 - Creating pull requests (processed later by maintainer)

Links



- Git documentation
 - <http://git-scm.com/doc>
- Mercurial
 - <http://www.mercurial-scm.org/>, <http://hgbook.red-bean.com/>
- Repository servers
 - <https://github.com/>
 - <https://bitbucket.org/>
 - <https://gitlab.com/>
- Tools
 - Git for Windows (<http://msysgit.github.io/>), TortoiseGit (Win), SmartGit (<http://www.syntevo.com/smartygit/>)
 - TortoiseHg (Mercurial GUI, Windows)
 - SourceTree (<https://www.sourcetreeapp.com/>, Git and Mercurial)

Homework



- Assignment
 - <https://d3s.mff.cuni.cz/files/teaching/nswi154/ukoly/>
- Deadline
 - 13.10.2020 / 14.10.2020

Software Building

(Sestavování aplikací)

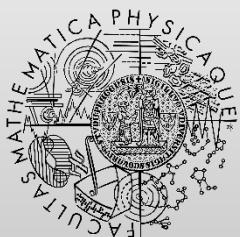
<http://d3s.mff.cuni.cz>

Department of
Distributed and
Dependable
Systems



Pavel Parízek

parizek@d3s.mff.cuni.cz



FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

What is software building



- Transforming source code (tree of files) into executable binary code
 - C/C++, C# → Win32 exe, Linux elf
 - Java, Scala → class files (bytecode)
- More generally
 - LaTeX source files → PDF documents

Requirements



- Automation
 - Minimal input from the developer
- Portability
 - Support for multiple platforms
- Efficiency
 - Process each source code file once
 - Reuse previously built fragments
- Robustness
 - Try building as much as possible
- Generality
 - Not only for a particular application
- Easy to use
 - How to easily define the build script

Challenges



- Dependencies
 - Building files in the correct order
 - first binary object files (.o) and then executable
 - Recompile after modification
 - header file (.h) → source code file (.c)
 - How to identify them properly
 - Pre-processor directives (“#include” in C)
 - Source code analysis (bytecode for Java)
 - Metadata and debug symbols in binaries
- Correct order
 - Logical dependencies between sources (.c, .java) and intermediate results (.o)

Other problems solved by build tools

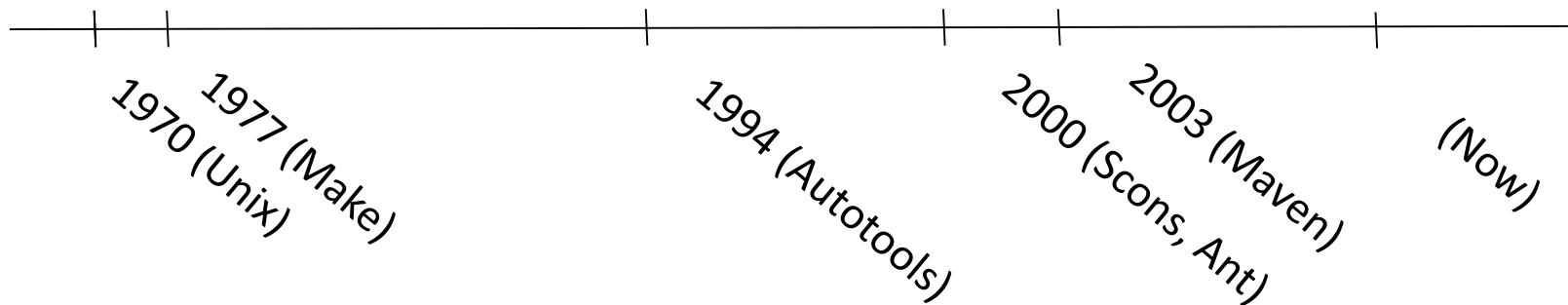


- Packaging (distribution)
 - generated binary files, metadata, documentation
- Running selected unit tests
- Generating documentation

Build tools



- Unix (C/C++): Make, Autotools
- Java world: Ant, Maven, Gradle
- Windows (.NET): MSBuild, (GUI)



Make



Make



- Standard build automation tool in the Unix and Linux world
- Used mainly for programs that use C/C++ and scripting languages (bash, Awk)
- Many derivatives exist
 - GNU Make, BSD Make, qmake, NMake, ...
- Build script: **Makefile**

Key concepts



- Target
 - Entity to be built: executable program, object file (.o), distribution package (.tgz)
 - Action to be done: clean, build all, prepare something
- Prerequisite
 - Entity that must be **available** and **up-to-date** before the associated target is fulfilled during the build process
- Rules
 - Dependencies between targets and prerequisites
 - Commands that fulfill targets (build entities, ...)

Makefile: example



target

all: progname

prerequisite

dependency

comment

progname: obj1.o obj2.o

<TAB> gcc -o progname obj1.o obj2.o

obj1.o: main.c config.h

gcc -c main.c

recipe

rule

Build process with Make



- Running
 - make *target*
 - make // default target
- Two steps
 - Construction of the build tree
 - Root node: target given by the user
 - Leaf nodes: available prerequisites
 - Processing rules in the tree

Task 1



- Download the “sockets” program
 - <http://d3s.mff.cuni.cz/files/teaching/nswi154/sockets.tgz>
 - Contains sources for a simple network client and server
 - Both have an UDP and TCP variant
 - Select using the parameter “–u”
- Write the Makefile for the “sockets” program
 - Useful targets: all, clean, program binaries, object files
 - Dependencies between entities (.o → binary, .c → .o, etc)
 - See the attached script build.sh for commands that can be used to compile source files
 - See clean.sh for commands to remove binaries and intermediate object files

Variables



```
objects = obj1.o obj2.o main.o \
          utils.o network.o gui.o
```

```
all : progname
```

```
progname: $(objects)
gcc -o prog $(objects) -lcommon
```

Note: wildcard expansion is quite tricky (manual, section 4.4)

Phony targets



- When the target does not represent any file

```
.PHONY : clean
```

```
clean :
```

```
    rm *.o
```

Guidelines



- Use built-in variables
 - CC // C compiler (gcc)
 - CFLAGS // C compiler flags
 - CXX // C++ compiler (g++)
 - CXXFLAGS // C++ compiler flags
 - ... and many more
- Use standard targets
 - all, clean, distclean, install

How to use built-in variables



- Define recipes properly

```
$ $(CC) $(CFLAGS) -c main.c
```

- Set flags when running Make

```
CFLAGS=-O2 make
```

Static pattern rules

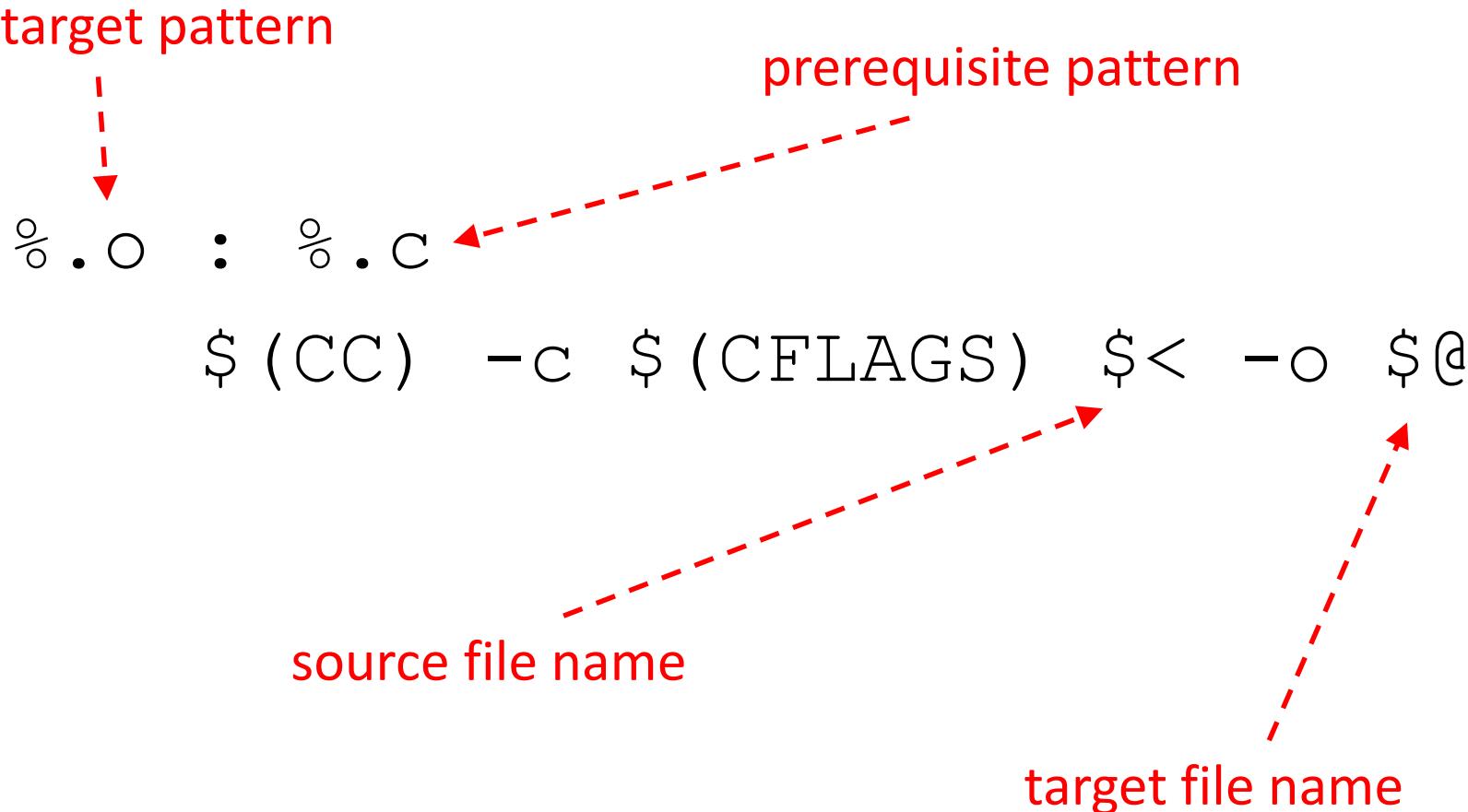


```
objects = main.o util.o network.o
```

```
$ (objects) : %.o : %.c
```

```
$(CC) -c $(CFLAGS) $< -o $@
```

Implicit rules



Task 2



- Rewrite the Makefile for “sockets”
- Eliminate duplication using these features:
 - Variables (built-in, custom)
 - Phony targets
 - Implicit rules
 - Static patterns
- Use dependencies between targets properly
- Respect common guidelines (best practices)

Recursive invocations (subdirectories)



```
SUBDIRS = src doc
```

```
.PHONY: subdirs $(SUBDIRS)
```

```
subdirs: $(SUBDIRS)
```

```
$(SUBDIRS):
```

```
$(MAKE) -C $@
```

Two flavors of variables



- Recursively expanded

```
objects = $(core_objs) $(server_objs)  
core_objs = tcp.o udp.o  
server_objs = srv/main.o
```

- Simply expanded

```
$ (core_objs) := tcp.o udp.o  
objects := $ (core_objs) srv/main.o
```

Substitutions



```
objects := main.o client.o server.o
```

```
sources := $(objects:.o=.c)
```

OR

```
sources := $(objects:%.o=% .c)
```

Operations with variables and values



- Appending

```
objects = main.o util.o  
objects += network.o
```

- Functions

```
$ (subst from, to, text)  
$ (patsubst pattern, replacement text)  
$ (filter pattern1 ... patternN, text)  
$ (dir path1 ... pathN)  
$ (basename path1 ... pathN)  
$ (suffix path1 ... pathN)
```

Automatic variables



- Target: \$@
- First prerequisite: \$<
- All prerequisites: \$^

Task 3



- Try more advanced features
 - Recursive invocation
 - Pattern substitutions
 - Text processing functions
 - Automatic variables

Other advanced features



- Order-only prerequisites
- Automated generating of files that capture prerequisites (suffix .d)
 - Good support by compilers
 - Fallback: makedepend tool
- Parallel execution
- Conditional directives
- ... and many more
- See the documentation for GNU Make

Limitations



- Portability over different Unix-like systems
 - Library functions in C (issues with compatibility)
 - Environment: shell, utilities (Awk, sed, grep, ...)
- Hard to maintain complex Makefiles
- Writing rules by hand can be tedious
- Solution: GNU build system (Autotools)
 - Tools: Autoconf, Automake, Libtool, gettext
 - De-facto standard in the open-source world
 - ./configure ; make ; make install

Links



- <http://www.gnu.org/software/make/>
- <http://www.gnu.org/software/make/manual/>
- <https://docs.microsoft.com/en-us/cpp/build/nmake-reference?view=vs-2017>

Homework



- Assignment
 - <http://d3s.mff.cuni.cz/files/teaching/nswi154/ukoly>
- Deadline
 - 20.10.2020 / 21.10.2020

Software Building

(Sestavování aplikací)

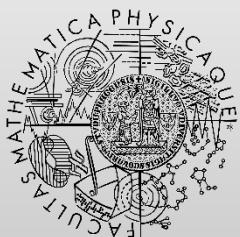
<http://d3s.mff.cuni.cz>

Department of
Distributed and
Dependable
Systems



Pavel Parízek

parizek@d3s.mff.cuni.cz



FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

Ant



Ant



- Build tool mostly for Java projects
 - Wide support of tools and frameworks common in the Java world (JUnit, JSP/Servlets, EJB, ...)
- Web: <http://ant.apache.org/>
- Highly extensible
 - Plug-ins written in Java
- Very portable scripting
- Scripts written in XML
 - Default file name: build.xml

Build file structure



```
<project name="MyProject" default="dist" basedir=".">
  <property name="src.dir" value="./src"/>
  <property name="build.dir" value="./build"/>
  
  <target name="init">
    <mkdir dir="${build.dir}"/>
  </target>
  
  <target name="compile" depends="init">
    <javac srcdir="${src.dir}" destdir="${build.dir}"/>
  </target>
  
  <target name="clean">
    <delete dir="${build.dir}"/>
  </target>
</project>
```

Terminology



- Task
 - Specific action to be performed during the build process
 - execute the Java compiler, create new directory
- Target
 - Goal required for building (compilation, packaging, running tests, generating documentation)
 - Set of tasks that must be executed to fulfill the goal
 - May have dependencies on other targets
- Project
 - Set of targets relevant for the application
- Property
 - name-value pair (strings)
 - usage: \${prop.name}

Dependencies between targets



- Build script

```
<target name="A"/>
<target name="B" depends="A"/>
<target name="C" depends="B"/>
<target name="D" depends="A,C"/>
<target name="E" depends="D,C,A"/>
```

- Execution order

E → D,C,A,E

D,C,A,E → A,C,D,C,A,E

A,C,D,C,A,E → A,B,C,D,C,A,E

A,B,C,D,C,A,E → A,A,B,C,D,C,A,E

A,A,B,C,D,C,A,E → A,A,B,C,D,A,B,C,A,E

A,A,B,C,D,A,B,C,A,E → A,B,C,D,E

Basic tasks



- Compilation of Java source files

```
<javac srcdir="${src.dir}" destdir=".build"
       debug="on" deprecation="on"/>
```

- Running an external Java program

```
<java classname="myapp.Main" fork="true">
  <arg value="nswi154"/>
  <jvmarg value="-Xmx512m"/>
</java>
```

- Packaging class files in JAR archive

```
<jar destfile="myapp.jar" basedir=".build">
  <manifest>
    <attribute name="Main-Class" value="..."/>
  </manifest>
</jar>
```

Online documentation



- <http://ant.apache.org/manual/index.html>
 - Important concepts (properties, filesets, paths)
 - List of core tasks (javac, java, file management)

Example 1



- Download the APISign package
 - <http://d3s.mff.cuni.cz/files/teaching/nswi154/apisign.tgz>
 - Contains two simple tools for generating and verifying file signatures
 - Taken from <http://docs.oracle.com/javase/tutorial/security/apisign/>
- Write the build script for APISign
 - Compilation
 - All source code files written in Java
 - Packaging
 - Create an archive `sign.jar` with the main class `GenSig`
 - Execution
 - Runs the archive `sign.jar` on a file name given as a command-line argument
 - Use properties where it makes sense, typical directory layout (`./src`, `./build`), and standard targets (`compile`, `build`, `init`, `clean`, `dist`)
 - Specify reasonable dependencies between targets
- Alternative (preferred): use your own programs instead of APISign
- Running Ant
 - Command-line: `ant <target name>`

File management



- Tasks

- <mkdir>
- <delete>
- <copy>
- <move>

Path-like structures



```
<path id="myapp.classpath">
    <pathelement path="${classpath}" />
    <fileset dir="lib">
        <include name="**/*.jar" />
    </fileset>
    <pathelement location="classes" />
    <dirset dir="${build.dir}">
        <include name="apps/**/classes" />
        <exclude name="apps/**/*Test*" />
    </dirset>
    <pathelement location="third_party/util.jar" />
</path>

<javac ...>
    <classpath refid="myapp.classpath" />
</javac>
```

Properties defined externally



- Create the file build.properties

```
src.dir=./src
```

```
build.dir=./build
```

```
lib.dir=./lib
```

- ... and include the file in build.xml

```
<property file="build.properties"/>
```

Example 2



- Download the CoCoME package
 - <http://d3s.mff.cuni.cz/files/teaching/nswi154/cocome.tgz>
 - It is a model of a trading system developed for research purposes
 - The package contains also few external libraries
- Alternative: use your own programs instead of CoCoME
- Write similar build.xml as in the first example
 - compilation, packaging, directory layout, target “clean”, properties, dependencies
- Improve the build script
 - Define classpath properly and use in tasks like <javac>
 - Use an external file to define properties

Dependencies between source files



- Recompile everything from scratch
 - We can probably recommend this approach
- Use task <depend>
 - Deletes all obsolete .class files (modified sources)
 - Re-use of some previously compiled class files
 - **Limitation: cannot discover some dependencies**
 - Example

```
<depend srccdir=".src" destdir="${build.dir}" />
```

Conditional processing of targets



- Property value is set/unset

```
<property name="my-cond" value="..."/>  
<target name="..." if="my-cond">  
<target name="..." unless="my-cond">
```

- File availability

```
<available file="./lib" property="have_lib"/>  
<target name="copy-libs" if="have_lib">  
    <copy ...>  
</target>
```

Boolean conditions



```
<condition property="config_debug">
  <and>
    <available file="../config"/>
    <or>
      <istrcmp value="debug_mode"/>
      <istrcmp value="testing"/>
    </or>
  </and>
</condition>
```

Explicit processing of targets

```
<target name="dist" depends="build-main,build-test">
  <antcall target="prepare-dist">
    <param name="release" value="2.1-rc3"/>
    <param name="website" value="http://..."/>
  </antcall>
</target>
```



Example 3



- Play with advanced features
 - Conditional processing of targets based on property values
 - Boolean conditions (setting properties)
 - Explicitly trigger processing of some target

Scripting



```
<parallel>
    task 1
    <sequential>
        task 2
        task 3
    </sequential>
    task 4
</parallel>
```

Scripting



- Loops
 - <http://d3s.mff.cuni.cz/files/teaching/nswi154/ant-contrib-1.0b3.jar>
 - Taken from: Ant-Contrib Tasks (<http://ant-contrib.sourceforge.net>)

- Example

```
<taskdef resource="net/sf/antcontrib/antlib.xml">
    <classpath>
        <pathelement location="ant-contrib-1.0b3.jar"/>
    </classpath>
</taskdef>

<for list="1,2,5,10" param="count">
    <sequential>
        <java classname="MyApp">
            <arg value="@{count}" />
        </java>
    </sequential>
</for>
```

Other useful tasks



- Executing arbitrary system commands

```
<exec executable="cmd.exe"  
       timeout="1000" output="log.txt"  
       errorproperty="error.msg">  
    <arg value="some_data"/>  
</exec>
```

- Creating archives: <zip>, <tar>
- Setting properties that contain the current date and time
`<tstamp/>`
- Printing messages
`<echo message="Error: ${error.msg}">`

Maven



Maven



- Project management and building tool
 - mainly for Java
- Typical usage scenarios made simpler for users
- Encourages best-practices and conventions
 - Directory layout
 - Naming of tests
- Web: <http://maven.apache.org/>

Best-practice guidelines



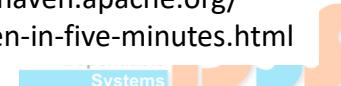
- Directory tree (layout)

```
my-app
  -- pom.xml
  -- src
    -- main
      -- java
        -- com
          -- mycompany
            -- app
              -- App.java
      -- resources
    -- test
      -- java
        -- com
          -- mycompany
            -- app
              -- AppTest.java
  -- target
    -- classes
```

- Test case names

**/*Test.java, **/Test*.java

Example taken from <http://maven.apache.org/guides/getting-started/maven-in-five-minutes.html>



Key concepts



- Goal
 - Single action to be executed
 - Construction of directory layout
 - Compilation of Java sources
 - Similar to **task** in Ant
- Phase
 - Step in the build lifecycle
 - generate-sources, compile, deploy
 - Sequence of goals
 - Similar to **target** in Ant
- Build lifecycle
 - Ordered sequence of phases
 - Similar to **dependencies between targets** in Ant

Typical build lifecycle



1. validate
 2. compile
 3. test
 4. package
 5. integration-test
 6. verify
 7. install
 8. deploy
- A red dashed line connects the "install" step to the text "to local repository" in red.

Project Object Model (POM)



- Project's configuration (build script)
 - Stored in the pom.xml file

```
<project>
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.mycompany.app</groupId>
  <artifactId>my-app</artifactId>
  <packaging>jar</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>Maven Quick Start Archetype</name>
  <url>http://maven.apache.org</url>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.8.2</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

Example taken from <http://maven.apache.org/guides/getting-started/maven-in-five-minutes.html>

Usage



- Project setup

```
mvn archetype:generate \
  -DarchetypeArtifactId=maven-archetype-quickstart \
  -DgroupId=com.mycompany.app -DartifactId=my-app
```

- Build lifecycle: mvn <*name of a phase*>

- Compilation: mvn compile
- Packaging: mvn package
- Web-site generation: mvn site
- Rebuild into local repository: mvn clean install

- Default remote repository (central)

- <http://repo1.maven.org/maven2/>

Advanced features



- Creating local repositories
- Creating packages with metadata
 - To be stored into repository
- Modifications of standard workflow
- Project inheritance (modules)
- Extensibility via plugins
 - Plugin implements a set of related goals

Example



- [http://d3s.mff.cuni.cz/files/teaching/nswi154/
maven-ex.tgz](http://d3s.mff.cuni.cz/files/teaching/nswi154/maven-ex.tgz)
 - DSI Utilities: original sources, build.xml, **pom.xml**
 - Project home page: <http://dsiutils.di.unimi.it/>

Want to know more about Maven ?



- Read the guide
 - <http://maven.apache.org/guides/>
- Try it yourself
 - Create new project
 - Add source files
 - Run compilation

Evaluation: Ant versus Maven



- Ant
 - Very flexible, gives you control over the build
 - Better for small/student projects (less overhead)
- Maven
 - Quite heavy, enforces lot of best practices
 - Good for large SW projects (enterprise-level)



- XML syntax of build scripts (“Makefiles”)
 - Used internally by Visual Studio 20xx-19
 - Syntax evolving (non-trivial differences)
 - Familiar concepts: task, target, property
-
- Homepage
 - <https://docs.microsoft.com/en-us/visualstudio/msbuild/msbuild?view=vs-2015>

Support for C# in Ant



- Plugin for Ant
 - <http://ant.apache.org/antlibs/dotnet/index.html>
- NAnt
 - <https://sourceforge.net/projects/nant/>
 - <http://nant.sourceforge.net/>

Homework



- Assignment
 - <http://d3s.mff.cuni.cz/files/teaching/nswi154/ukoly>
- Deadline
 - 3.11.2020 / 4.11.2020
- First part assumes usage of Ant by default
 - Option: use MSBuild on FTP server
 - Find how to invoke the Java compiler
- Alternative: create build.xml script for your C# program of reasonable size (“zápočták”)

Software Building

(Sestavování aplikací)

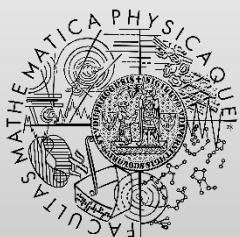
<http://d3s.mff.cuni.cz>

Department of
Distributed and
Dependable
Systems



Pavel Parízek

parizek@d3s.mff.cuni.cz



FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

Outline

- NuGet
- Gradle
- GNU build system
- CMake



- Package manager for .NET
- Similar concepts to Maven
- Integration to Visual Studio
- Web: <https://www.nuget.org/>
- Docs: <https://docs.microsoft.com/en-us/nuget/>

.NET Core Templates



- Support for project templates
 - Implementing best & recommended practices
- Additional information
 - <https://docs.microsoft.com/en-us/dotnet/core/tools/custom-templates>
 - <https://docs.microsoft.com/en-us/dotnet/core/tools/dotnet-new>
- Template engine
 - <https://github.com/dotnettemplating/>
- Available templates
 - <https://github.com/dotnettemplating/wiki/Available-templates-for-dotnet-new>

Gradle



- Another popular general-purpose build tool
 - Java, Scala, C, C++, Android
 - Encourages best practices (like Maven)
 - Script language (DSL) based on Groovy
-
- Web: <https://gradle.org/>

Gradle – examples



- Scripts

- https://docs.gradle.org/current/userguide/tutorial_using_tasks.html
- https://docs.gradle.org/current/userguide/building_java_projects.html

- Running

- gradle clean build
- gradle run

- Project template for Java

- gradle init --type java-application

Motivation for GNU build system



- Portability of programs
 - over different UNIX-like systems
 - existing standards (C, POSIX) define only core aspects
- System-specific configuration
 - e.g., use of KDE instead of Gnome
- Complexity of Make files
 - unreadable, hard to maintain
 - writing all the rules is tedious
- Portability of Make files
 - Make is standardized by POSIX, but not all UNIX-like systems are 100% compliant

Selected portability and compatibility issues



- Programs in C
 - `exit()`: may return `void` or `int` (error code)
 - `free(NULL)`: sometimes does nothing
 - `malloc(0)`: returns `NULL` or valid pointer
 - (and many more)
- Functions in different headers and libraries
- Shell and utilities: Awk, Grep, Sed, ...
 - Multiple implementations (not all compatible)

Solutions for portability and compatibility



- Virtualized environment (Java, C#/.NET)
- GNU build system (Autotools)
 - **De-facto standard in Unix/Linux world**
 - Explicit support for different flavors
 - Database of known portability issues
 - Resolves issues during configuration
 - Uses only features available everywhere

GNU build system (Autotools)



- Autoconf
 - Configuration detector
- Automake
 - Makefile generator
- Libtool
 - Abstracts creation of libraries
- Gettext
 - Support for localization

End user's perspective



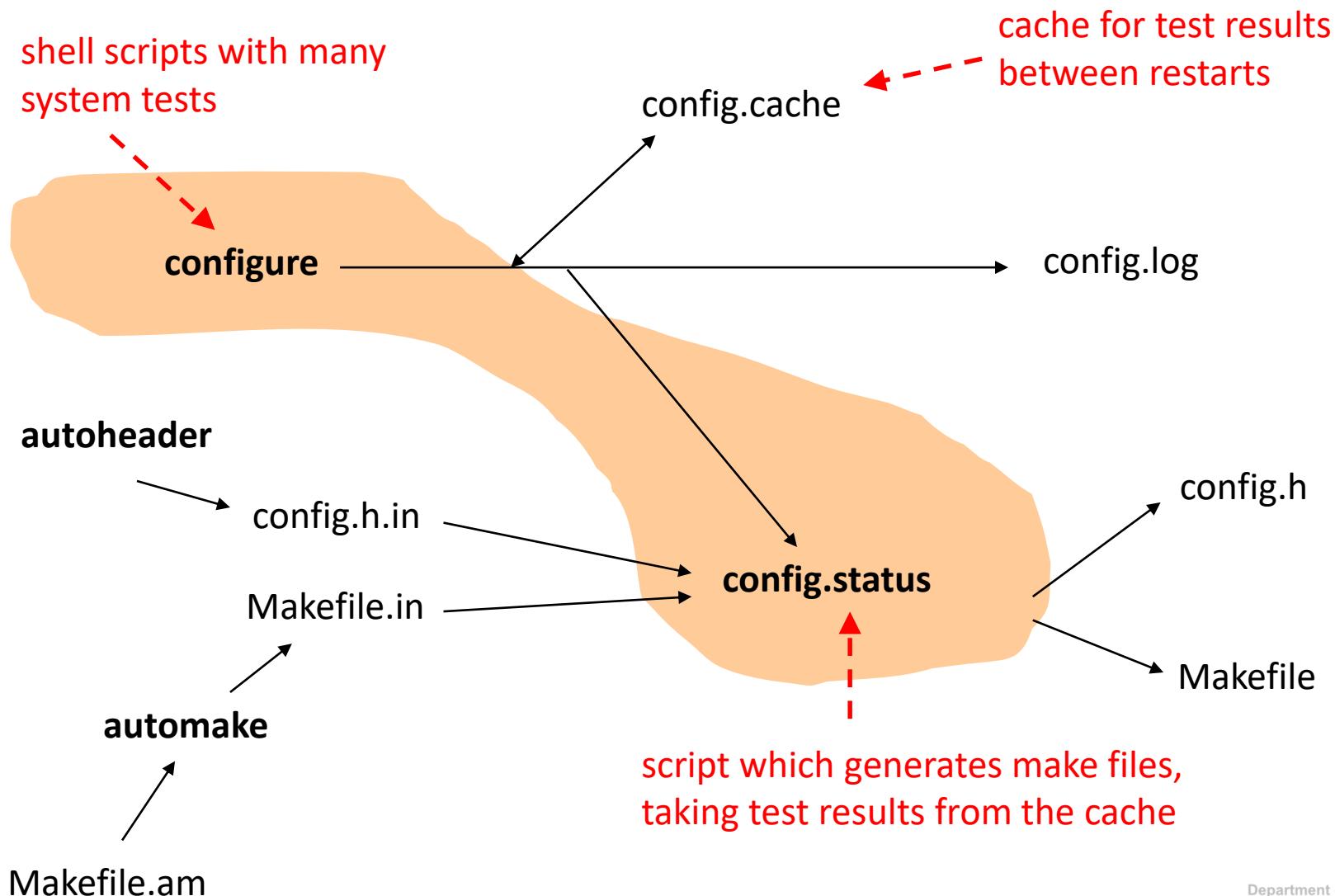
1. Download the source code
2. “./configure”
 - Automatically tests the target system
 - e.g. for presence of required libraries
 - Detects system configuration (OS, HW)
 - Automatically generates Make files
3. “make”
4. “make install”

End user's perspective – configuration

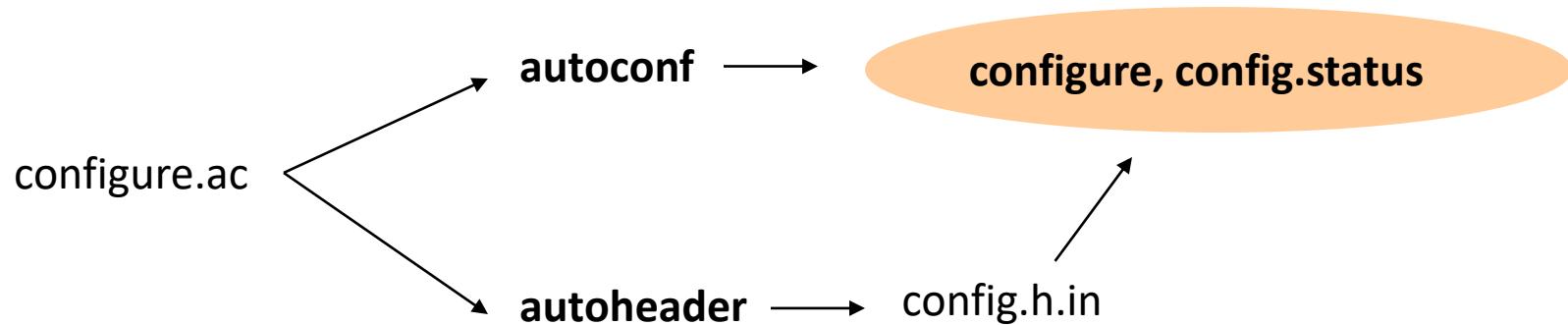


- Installation root directory
 - “configure --prefix=/opt”
- Cross-compilation
 - “configure --host”
- Optional features of the software
 - “configure --enable-FEATURE”
 - “configure --disable-FEATURE”
- Optional packages (libraries) to build with
 - “configure --with-PACKAGE”
 - “configure --without-PACKAGE”

What is behind the scenes



Autoconf & the “configure” script



- Very portable shell script
 - Uses features in the lowest-common-denominator of known shells (no functions, ...)
 - Generated from a template (**configure.ac**)
 - Based on a library of tests of well-known portability and compatibility issues

“configure.ac” script template



AC_INIT(package, version, bug-report-address)

information about the package

checks for programs

checks for libraries

checks for header files

checks for types

checks for structures

checks for compiler characteristics

checks for library functions

checks for system services

AC_CONFIG_FILES([output file, ...])

AC_OUTPUT

“configure.ac” – example

```
AC_INIT([GNU cflow], [1.2], [bug-cflow@gnu.org])
```

```
AC_CONFIG_HEADER([config.h])
```

Checks for programs.

```
AC_PROG_CC
```

```
AC_PROG_LEX
```

Checks for header files.

```
AC_HEADER_STDC
```

```
AC_CHECK_HEADERS([stdlib.h string.h unistd.h locale.h])
```

```
AC_OUTPUT
```

multiple arguments
to a macro

macros

a single argument to a macro

“configure” – another example



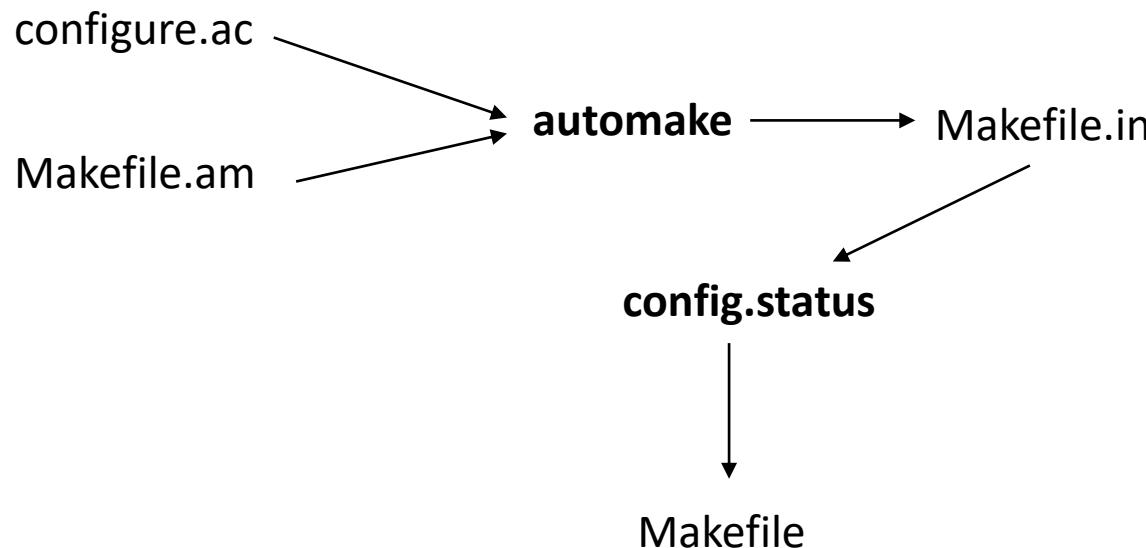
- [http://d3s.mff.cuni.cz/files/teaching/nswi154/
sockets-auto.tgz](http://d3s.mff.cuni.cz/files/teaching/nswi154/sockets-auto.tgz)
 - configure.ac
 - src/config.h.in
 - src/client.cpp
- Achieving support for multiple platforms

Generating “configure.ac”



- Autoscans
 - Inspects source files (C/C++) to detect common portability issues
 - Generates skeleton of configure.ac
- Ifnames
 - Reports variables used in preprocessor conditionals
 - Often used to solve platform dependency issues
 - Example: #if HAVE_LOCALE_H

Automake – creating portable Makefiles



Supported targets



- install, install-exec, install-data
- uninstall
- clean
- distclean
 - clean to what is distributed
 - removes also files generated by configure
- check
 - run test of compiled binaries
- installcheck
 - run test of installed program
- dist
 - creates source code distribution package (tarball)

“Makefile.am” template



Makefile.am

```
SUBDIRS = src  
dist_doc_DATA = README
```

directories to be processed
before this directory

install README into docdir
and put it into distribution

src/Makefile.am

```
bin_PROGRAMS = hello  
hello_SOURCES = main.c
```

“Makefile.am” template



Makefile.am

```
SUBDIRS = src  
dist_doc_DATA = README
```

“hello” is a program to
be installed into bindir

program “hello”
can be built from
source “main.c”

src/Makefile.am

```
bin_PROGRAMS = hello  
hello_SOURCES = main.c
```

since “main.c” is a source file, it will also
be put into distribution (by “make dist”)

Automake – creating libraries



- Static libraries (.a)

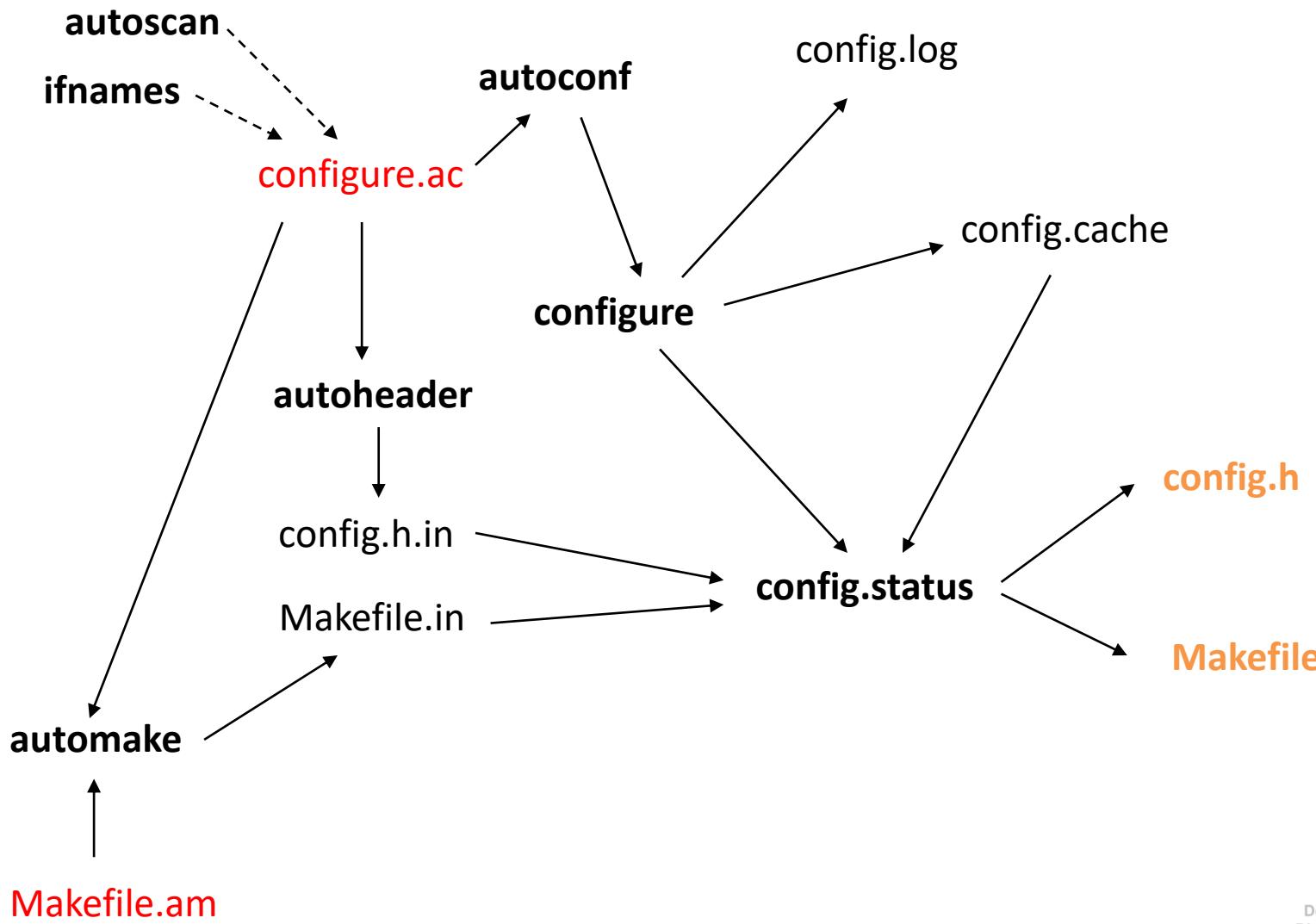
```
noinst_LIBRARIES = libgreeting.a
libgreeting_a_SOURCES = greet.c ...
```

```
bin_PROGRAMS = hello
hello_SOURCES = main.c
hello_LDADD = libgreeting.a
```

- Shared libraries (.so)

- https://www.gnu.org/software/automake/manual/html_node/A-Shared-Library.html

Autotools – the whole picture again



Further reading



- <http://www.sourceware.org/autobook/>
- <https://www.gnu.org/software/autoconf/>
- <https://www.gnu.org/software/automake/>

CMake



- Cross-platform free and open-source build management application
- Compiler-independent tool
 - Supports various native build systems (make, Xcode, MS Visual Studio)
- Web: <http://www.cmake.org/>
- Two phases of the build process
 - Generate native build scripts from platform-independent configuration (CMakeLists.txt)
 - Run target platform's native tool for the actual build

Other build tools



- Ivy
 - <https://ant.apache.org/ivy/>
- Scons
 - <http://www.scons.org/>
- Bazel
 - <http://bazel.io/>

Homework



- Assignment
 - <http://d3s.mff.cuni.cz/files/teaching/nswi154/ukoly>
- Deadline
 - 10.11.2020 / 11.11.2020

Functional Testing

(Testování funkčnosti)

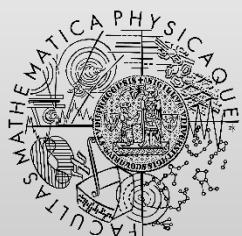
<http://d3s.mff.cuni.cz>

Department of
Distributed and
Dependable
Systems



Pavel Parízek

parizek@d3s.mff.cuni.cz



FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

Software testing



- Purpose
 - Checking whether a given program satisfies certain requirements and expectations about its behavior
- Basic idea
 - Pick specific inputs (a set of values)
 - Run the program for each input
 - Inspect the output and final state
- Shows only presence of errors
 - You can try just few selected input values

Terminology



- Test case
 - Checks single requirement on the program behavior
 - Defines test input and expected output (final state)
- Test suite
 - Collection of related test cases
- Fixture
 - Common environment for test cases in a given suite

When to run tests



- Development
 - 1) Write code and some tests
 - 2) Run all tests and find bugs
 - 3) Fix bugs detected by tests
 - 4) Go to step 1 until deadline
- Regressions
 - Execute all passed tests after every modification
 - bug fix, refactoring, new unrelated feature, optimization
 - Goal: check whether everything still works then

Testing on different levels



- **Unit testing**
 - Small components (method, class)
 - Automatic easily repeatable tests
 - Provides clear answer (pass or fail)
- **Integration testing**
 - Checking interaction between components
- **System testing**
 - Whole system in a target environment
 - Requirements specified by customers

Unit testing



- Developers write code that
 - Specifies test inputs and required properties
 - Checks whether all tests successfully passed
 - Comparing expected outputs (and program state) with actual outputs
- Frameworks
 - **JUnit**, PyUnit, CPPUNIT, NUnit, xUnit, MSTest, ...



- Unit testing framework for Java
 - <https://github.com/junit-team/junit/wiki>
 - <http://junit.org/junit5/>
- Key features
 - Test cases are normal Java methods
 - Test suites are normal Java classes
 - Results analyzed in an automated way
- Versions
 - JUnit 3.8.x: fixed method names, reflection
 - **JUnit 4.x/5: annotations**

Simple test case



```
import java.util.*;  
  
import org.junit.Test;  
import static org.junit.Assert.*;  
  
public class TestArrayList {  
    @Test  
    public void add() {  
        List al = new ArrayList();  
        int origSz = al.size();  
        al.add("abc");  
        int newSz = al.size();  
        assertEquals("new != orig+1", origSz+1, newSz);  
        assertTrue(al.contains("abc"));  
    }  
}
```

Assert statements



- `public static void assertXY ([message], ...)`
- `assertEquals(T expected, T actual)`
- `assertArrayEquals(T[] expected, T[] actual)`
- `assertSame(Object expected, Object actual)`
- `assertTrue(boolean condition)`
- `assertFalse(boolean condition)`
- `assertNull(Object obj)`
- `assertNotNull(Object obj)`
- `fail([String message])`

Running tests



- Command line

```
java -cp lib/junit-4.11.jar:<dir with tests>
      org.junit.runner.JUnitCore <test class name>
```

- Ant

```
<target name="run.tests" depends="build.tests">
  <junit haltonfailure="no">
    <formatter type="brief" usefile="false"/>
    <classpath refid="cp.run.tests"/>
    <batchtest>
      <fileset dir="${build.dir}">
        <include name="**/Test*.class"/>
      </fileset>
    </batchtest>
  </junit>
</target>
```

What you should test



- Method contracts (API)
- All branches in the code
- All control-flow paths
- Special (corner) cases
 - “off by one”, bad inputs
- Regressions
 - Inputs triggering previously discovered bugs

Task 1



- Write unit tests for `java.util.ArrayList`
 - Selected methods: `add(o)`, `get(i)`, `remove(i)`, `remove(o)`, `clear()`, `size()`, `contains(o)`
- Try different assert statements
- Create also some failing tests
 - Inspect output of JUnit to see how it typically looks
- JUnit library
 - <http://d3s.mff.cuni.cz/files/teaching/nswi154/junit-4.11.jar>
 - <http://d3s.mff.cuni.cz/files/teaching/nswi154/hamcrest-core-1.3.jar>
- C#/.NET variant
 - `ArrayList` from the namespace `System.Collections`
 - `List<T>` from `System.Collections.Generic`

Fixture



- Goal: prepare objects in a known state
 - Set up a fixed environment for each test cases
- Reset before each test case → isolated tests
- Initialization
 - @Before
 - @BeforeClass
- Clean-up
 - @After
 - @AfterClass

Test case with a simple fixture



```
import org.junit.*;  
  
public class TestArrayList {  
    private List al;  
  
    @Before  
    public void setUp() {  
        al = new ArrayList();  
        al.add("abc");  
    }  
  
    @After  
    public void tearDown() {  
        al = null;  
    }  
  
    @Test  
    public void add() { ... }  
}
```

Expected exceptions



```
@Test(expected=MyEx.class)
```

```
public void testSomething() {  
    doSomeOperationThatThrowsException();  
}
```

Task 2

- Extend your tests for ArrayList
- Define common fixtures
 - Extract duplicate initialization code
- Test against expected exceptions
 - get(i): IndexOutOfBoundsException

Recommended practice



- Place tests in the same package as target classes
 - Directory layout

```
src/main/cz/cuni/mff/myapp/MyClass.java  
src/tests/cz/cuni/mff/myapp/Test MyClass.java
```
- Define single assertion in each test method
 - JUnit reports only the first failed assert in a test case
 - Multiple assertions → some failures possibly missed
 - Drawback: you need to write/produce lot more code

Parameterized tests



```
@RunWith(Parameterized.class)
public class TestSquareRoot {

    @Parameters
    public static Collection<Object[]> data() {
        return Arrays.asList(new Object[][]{ {1,1}, {4,2} });
    }

    private int valInput;
    private int expOutput;

    public TestSquareRoot(int i, int e) {
        valInput = i; expOutput = e;
    }

    @Test
    public void test() {
        assertEquals(expOutput, Math.sqrt(valInput));
    }
}
```

HTML report



```
<target name="test">
  <junit fork="on">
    <b><formatter type="xml" /></b>
    <classpath refid="....">
      <batchtest>
        ...
      </batchtest>
    </junit>
  </target>

<target name="report">
  <b><mkdir dir="reports"/></b>
  <junitreport todir=". /reports">
    <fileset dir=". . ">
      <include name="TEST-* .xml" />
    </fileset>
    <report format="frames" todir=". /reports/html" />
  </junitreport>
</target>
```

Task 3

- Use some parameterized tests
- Try reports in HTML (with Ant)

Advanced features of JUnit



- Matchers
 - `assertThat`
- Assumptions
- Rules
 - `TemporaryFolder`
 - `ErrorCollector`
- Categories
- Further information
 - <https://github.com/junit-team/junit/wiki>

JUnit 5 – new features



- Framework decomposed into several modules
- Distributed through Maven central repository
- User guide
 - <https://junit.org/junit5/docs/current/user-guide/>
- New syntax of annotations
 - @Before vs @BeforeEach, @After vs @AfterEach
 - @BeforeClass vs @BeforeAll, @AfterAll
- New modern API
 - Classes and interfaces => different imports
 - Named assertions, grouping via assertAll
 - Syntax for parameterized tests (data source)

Testing methods



- Black-box testing
 - Zero knowledge about the implementation (no access)
 - Tests based only on specification and interfaces (API)
 - Checking outputs against expectations for input values
- White-box testing
 - Full knowledge of the implementation (access to code)
 - Tester can modify the system a little bit for easy testing
- Grey-box testing
 - Tester knows the system (code), but cannot modify it

Dependencies among objects



- Units typically have dependencies
 - Very hard to test such units in full isolation
 - Approach: complex fixtures and test cases
 - Example

```
@Before  
public void setUp() {  
    java.sql.Connection db = ... // complex init  
    PersistenceMngr pm = new MyPersistenceMngr(db);  
}
```

- Possible solutions
 - dummy objects, fake, stubs, mock objects

Dependencies among objects



- Dummy objects
 - Passed around but never used (e.g., parameter list)
- Fake
 - Working simpler implementation (e.g., in-memory DB)
- Stub
 - “empty” implementation with predefined responses to method calls
- Mock object
 - Stub that also checks whether it is used correctly by the object under test → “behavior verification”
 - Frameworks: EasyMock, Mockito, Rhino Mocks, Moq

Concurrency



- Testing does not work for concurrency
 - Programs with multiple threads
- Huge number of thread schedules
- Non-deterministic behavior
- Errors are hard to reproduce

Unit testing for Windows/.NET



- MSTest (Visual Studio)
 - Annotations: [TestClass], [TestMethod]
 - Basic assertion statements
 - Assert.AreEqual(Object, Object, String)
 - IsTrue, IsNotNull, IsInstanceOfType, Fail, ...
 - More advanced: StringAssert, CollectionAssert
- Other frameworks
 - NUnit: <http://nunit.org/>, <https://github.com/nunit>
 - xUnit.net: <http://xunit.github.io/>

Automation



- Generating tests with dynamic symbolic analysis
 - Manual writing of tests is very tedious
 - KLEE: <http://klee.github.io/>
 - IntelliTest: <https://docs.microsoft.com/en-us/visualstudio/test/generate-unit-tests-for-your-code-with-intellitest?view=vs-2017>
- Fuzzing techniques and tools
 - Search for inputs that may trigger some errors
 - SAGE & DART
 - Information and links: <https://patricegodefroid.github.io/>
 - JDart: <https://github.com/psycopaths/jdart>
 - Useful for security bugs (critically important, hard-to-find)

Related courses



- More general information about testing
 - NTIN070: Testování software (ZS)
- But you can do better than simple unit testing ...
 - **NSWI126: Pokročilé nástroje pro vývoj a monitorování software (LS)**
- ... and you can even model, analyze, and verify program behavior
 - NSWI101: Modely a verifikace chování systémů (ZS)
 - **NSWI132: Analýza programů a verifikace kódu (LS)**

Links



- JUnit
 - <https://github.com/junit-team/junit/wiki>
 - <http://junit.org/junit5/>
- MSTest
 - <https://docs.microsoft.com/en-us/visualstudio/test/unit-test-your-code?view=vs-2017>
- NUnit
 - <http://www.nunit.org>
 - <https://github.com/nunit/docs/wiki/NUnit-Documentation>
- CPPUNIT
 - <http://sourceforge.net/projects/cppunit>
- Catch2
 - <https://github.com/catchorg/Catch2>
- Google Test
 - <https://github.com/google/googletest>

Homework



- Assignment
 - <http://d3s.mff.cuni.cz/files/teaching/nswi154/ukoly/>
- Deadline
 - 24.11.2020 / 25.11.2020
- Homework targets Java and JUnit
 - Alternative 1: C# and suitable framework
 - They use same concepts but little bit different syntax
 - Write similar test cases for the corresponding classes from the .NET base class library (e.g., SortedDictionary)
 - Alternative 2: In fact, any other language with support for unit testing can be used
 - For example: C++, Python, Scala

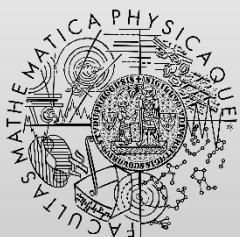
Debugging & Bug-finding

<http://d3s.mff.cuni.cz>



Pavel Parízek

parizek@d3s.mff.cuni.cz



FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

Motivation



- When some test fails
 - You know there is a bug in the program code
 - You **do not** know the root cause of the bug
- Testing detects presence of bugs in the code
 - But you still have to find them and eliminate properly
 - Writing tests for smaller units of code does not help
 - Too much work with a little benefit (bad “cost-effect” ratio)
- Solution: debugging, automated bug-finders

Debugging



- Manual process
 - Monitoring execution of a given program
 - Inspecting and updating the current state
- Tool support
 - Stop and restart program execution
 - Manage breakpoints (set, delete)
 - Inspect and update memory content
 - e.g., the current values of program variables
 - Attach debugger to a running program

Important concepts



- Breakpoint
 - Source code location where the program execution is stopped intentionally
 - Additional conditions may have to be also satisfied
 - total number of hits, the current value of a program variable
 - Types: HW (CPU, fast, limited), SW (interrupt, slow)
- Core dump
 - Full memory image of the crashed process
 - heap objects and fields, registers, stack trace of each thread
 - Records the full program state upon crash

Basic approaches



- Printing debug messages
 - Add many `print` statements into your code
 - `System.out.println("[DEBUG] MyObj.doSmth: arg1 = " + arg1 + ", v = " + v + ", data = " + this.data);`
 - Read huge log files (search for text patterns)
 - Useful when you need lot of data at the same time
- “Online” debuggers
 - Control program execution and inspect current state
 - Basic tools: **GDB**, DDD, jdb, JPDA, WinDbg, KD, CDB
 - IDE support: Visual Studio, Eclipse, NetBeans, IDEA
- Thorough explanation of your code to friends/colleagues
 - Approach works surprisingly well in practice

The [complete] process of debugging



Debugging bothers
more than coding...

Six Steps of Debugging

1. That can't happen.
2. That doesn't happen on my machine.
3. That shouldn't happen.
4. Why does that happen?
5. Oh, I see.
6. How did it ever work???

GNU Debugger (GDB)



GNU Debugger (GDB)



- User interface: command-line
- Intended for Unix-like systems
 - Low-level system software written in C/C++
 - Examples: utilities, web server, operating system kernel
- Supports many languages
 - C, C++, Ada, Pascal, Objective-C, ...
- Web site
 - <http://www.sourceware.org/gdb/>

Running program with GDB



- Start GDB for a given program

```
gdb <program>
```

- Start program with arguments

```
gdb --args <program> <arg1> ... <argN>
```

- Run program again inside GDB

```
(gdb) run [<arg1> ... <argN>]
```

- Exit the debugged program

```
Ctrl+d (EOF)
```

- End the GDB session

```
(gdb) quit
```

Breakpoints



- Define breakpoint

(gdb) break <function name>

(gdb) break <line number>

(gdb) break <filename>:<line>

- Continue execution

(gdb) continue

- Shortcut: (gdb) c

Breakpoints



- List of breakpoints

(gdb) info breakpoints

- Disable breakpoint

(gdb) disable <num>

- Enable breakpoint

(gdb) enable <num>

- Delete breakpoint

(gdb) delete <num>

Single stepping



- Advance to the next source line

(gdb) step [count]

- Shortcut: (gdb) s

- Advance to the next line in the current scope

(gdb) next [count]

- Shortcut: (gdb) n

Information about the debugged program



- Source code lines

```
(gdb) list
```

```
(gdb) list <linenum>
```

- Symbol table

```
(gdb) info scope <function name>
```

```
(gdb) info source
```

```
(gdb) info functions
```

```
(gdb) info variables
```

```
(gdb) info locals
```

Information about program variables



- Values

(gdb) print <expression>

- Example: (gdb) print argv[1]
- Shortcut: (gdb) p

- Types

(gdb) whatis <variable name>

(gdb) ptype <variable name>

Inspecting the call stack frames



- Print call stack

- (gdb) backtrace

- Shortcut: (gdb) bt
 - Including local variables

- (gdb) bt full

- Selecting frames

- Move frame up: (gdb) up [n]
 - Move down: (gdb) down [n]

Changing expression values



- Make changes

```
(gdb) set var <expr> = <new value>
```

```
(gdb) print <expr> = <new value>
```

- Watch for changes (data breakpoint)

```
(gdb) watch <expression>
```

- List all watchpoints

```
(gdb) info watchpoints
```

Core dumps



- Set maximum size of core files

```
ulimit -c unlimited
```

- Analyze the core dump file (“core”)

```
gdb <program binary> <core dump>
```

- Attach to already running process

```
gdb <program binary> <process ID>
```

Advanced features of GDB



- Calling functions and jumps
- Breakpoint command list
- Support for multi-threading
- Reverse execution
- Record and replay
- Remote debugging
- GUI frontend: DDD
 - <http://www.gnu.org/software/ddd>

Concurrency



- Debuggers support multi-threaded programs
 - Including GDB
- Problems
 - Programs behave differently when running in the debugger than in normal execution
 - Different internal timing of concurrent events
 - It is hard to find concurrency bugs with debuggers

Debugging tools for Windows/.NET



- Visual Studio debugger
 - Supported languages: C#, Visual Basic, ASP .NET
 - Advanced features: edit & continue, attach to running process, scriptability
 - No support for debugging kernel space code
- Other tools
 - Windows debuggers (Windows SDK, WDK)
 - <https://docs.microsoft.com/en-us/windows-hardware/drivers/debugger/index>
 - <https://docs.microsoft.com/en-us/windows-hardware/drivers/debugger/debugger-operation-win8>
 - Tools: WinDbg, KD, CDB, Psscor4, various utilities
- GDB-based: Visual Studio GDB Debugger, Visual GDB

Task 1



- Example
 - <http://d3s.mff.cuni.cz/files/teaching/nswi154/sudoku.tgz>
 - Build with Make (sets flags “-g -Wall -O0”)
 - Run via the command ./sudoku vstup.txt
- Try basic features
 - Running the program in debugger
 - Management of breakpoints
 - Single stepping commands
 - Printing information about the program and variables
 - Inspecting the call stack and switching frames
 - Changing values of selected program variables

Automated run-time checking



- Idea: search for bugs during program execution
- Main approaches
 - Replacing libraries with debugging versions
 - Program linked with special versions of some library functions
 - Library functions (`malloc`, `free`, ...) perform runtime checks
 - Force program to crash upon a detected memory access error
 - Supported errors: buffer overflows, leaks, using freed memory
 - Tools: Dmalloc, DUMA
 - Monitoring execution of an instrumented program and looking for specific errors
 - Tools: **Valgrind**

Valgrind



- Generic framework for creating runtime checkers (error detectors)
 - Supported platforms
 - Linux: x86, x86-64, PowerPC
 - Android (x86, ARM), OS X
 - Basic principle: dynamic binary instrumentation
- Includes several tools
 - **MemCheck**: detects memory management errors
 - Helgrind: detects errors in thread synchronization

Running



- Command line:

```
valgrind <program> <arguments>
```

- Recommended compiler flags to use

```
-g -O0 -Wall -fno-inline
```

- Avoid optimizations (-O1,-O2) when using Valgrind to detect errors in your program

MemCheck



- Running

```
valgrind [--tool=memcheck] <program>
```

- Supported errors

- Accessing freed memory blocks
- Reading uninitialized variables
- Double-freeing of heap blocks
- Memory leaks (missing “free”)

- How to enable leak detection

```
valgrind --leak-check=yes <program>
```

MemCheck: output



- Buffer overflow

PID

```
== 2456 == [Invalid write of size 4]
== 2456 ==      at 0x204A68D: myfunc (myprog.c:95)
== 2456 ==      at 0x204A120: main (myprog.c:14)
== 2456 == Address 0x2684FF0 is 8 bytes after a block of
                size 64 alloc'd
== 2456 ==      at 0x2684FA8: malloc (vg_replace_malloc.c:130)
== 2456 ==      by 0x204A0E8: main (myprog.c:10)
```

kind of error

stacktrace
identifies the
point where the
error occurred

description of the memory address
involved in the error

- Memory leak

```
== 1789 == 32 bytes in 1 blocks are definitely lost in loss
            record 1 of 1
== 1789 ==      at 0x2F4482D: malloc (vg_replace_malloc.c:130)
== 1789 ==      at 0x204A692: myfunc (myprog.c:112)
== 1789 ==      at 0x204A130: main (myprog.c:20)
```

Issues



- Performance
 - Instrumented program runs 5-30 times slower than normal and uses much more memory
- Missed errors
 - Cannot detect off-by-one errors in the use of data allocated statically or on the stack
- Optimizations
 - Does not work well with -O1 and -O2

Task 2



- Try using MemCheck on the sudoku program
 - Inspect reported warnings (memory leaks)
- Try using Valgrind on some programs in the Linux distribution (`ls`, `cat`, ...) and on your simple programs in C/C++

Advanced topics



- Suppressions
 - Ignoring reported false positives and errors found in system libraries
- Useful options
 - read-var-info=yes
 - Information about variables (name, type, location)
 - track-origins=yes
 - Shows where the uninitialized variables come from
- Connecting Valgrind with GDB

Links



- GDB
 - <http://www.sourceware.org/gdb>
- jdb: The Java Debugger
 - <http://docs.oracle.com/javase/8/docs/technotes/tools/unix/jdb.html>
- Dmalloc
 - <http://dmalloc.com>
- DUMA
 - <http://sourceforge.net/projects/duma>
- Valgrind
 - <http://valgrind.org/>
- Sanitizers from Google (address, memory, leak, thread)
 - <https://github.com/google/sanitizers>

Static code analyzers



- Automated search for common problems in source code at compile-time
 - bug patterns, suspicious constructs, bad practice
- Focus on semantics (behavior)
 - Compiler has already checked the syntax
- Modular analysis (each procedure separately)
- Trade-off: precision versus performance
 - **false positives, missed errors**
- Detect only simple bugs in the source code
 - but still very useful (highly recommended to use)

What the analyzers detect



- Basic patterns
 - Possible null dereferences
 - Comparing strings with ==
 - Ignoring result of method call
 - Example: `InputStream.read()`
 - Array index out of bounds
- Wrong usage of API
 - Stream not closed when exception occurs
- Memory usage errors
 - `double free()`, possible leaks

Tools



- Java
 - **FindBugs**, Jlint, PMD, Error Prone, Checker Framework
- C/C++
 - **Clang**, PREfast, Cppcheck
- C#/.NET
 - StyleCop, FxCop, ReSharper, **Roslynator**

FindBugs



- Bug patterns detector for Java
- Source code available (LGPL)
- Usage: command line, GUI, Ant, Maven
- Integration with Eclipse (plugin)
- <http://findbugs.sourceforge.net/>

Demo: FindBugs



FindBugs: advanced features



- Filtering bugs
- Annotations
- Data mining

Clang static analyzer



- LLVM compiler infrastructure project
- Clang front-end (C, C++, Objective-C)
- Source code available (BSD-like license)
- User interface: command-line
- <http://clang-analyzer.llvm.org/>

Demo: Clang



- Command: `scan-build`
 - Intercepts standard build process (CC, CXX)
 - Runs compiler and then static code analyzer
- How to use it
 - `scan-build <your build command>`
 - Examples
 - `scan-build ./configure ; make`
 - `scan-build gcc test.c mylib.c`
- Output: HTML files (bug reports)

Clang: options



- List all available checkers
 - Command: `scan-build -h`
- Enabling some checker
 - `scan-build -enable-checker [name]`

Task 3



- FindBugs
 - Download and unpack
 - <http://findbugs.sourceforge.net/downloads.html>
 - How to run it
 - Linux/Windows: bin/findbugs
 - Other options (e.g., heap size)
 - <http://findbugs.sourceforge.net/manual/running.html>
- Clang static analyzer
- Target programs
 - Your own (“zápočťáky”, “softwarový projekt”)
 - Widely known open source software packages

Related courses



- Tools for detecting complicated bugs
 - concurrency (deadlocks, data races), assertions
 - NSWI101: Modely a verifikace chování systémů
 - **NSWI132: Analýza programů a verifikace kódu**

Links (other tools)



- Cppcheck
 - <http://cppcheck.sourceforge.net/>
- PMD
 - <http://pmd.github.io/>
- Error Prone
 - <http://errorprone.info/>
- FxCop
 - [https://docs.microsoft.com/en-us/previous-versions/dotnet/netframework-3.0/bb429476\(v=vs.80\)](https://docs.microsoft.com/en-us/previous-versions/dotnet/netframework-3.0/bb429476(v=vs.80))
- ReSharper
 - <https://www.jetbrains.com/resharper/>

Roslynator



- Extensible static analysis tool for C#
- Additional information
 - <https://www.infoq.com/news/2020/01/roslynator-analyzers-231/>
 - <https://github.com/JosefPihrt/Roslynator>
 - <https://devblogs.microsoft.com/dotnet/write-better-code-faster-with-roslyn-analyzers/>
 - <https://docs.microsoft.com/en-gb/visualstudio/code-quality/roslyn-analyzers-overview?view=vs-2019>

Checker Framework



- Extends type system of Java
- Source code annotations
- Compiler plugins (“checkers”)
 - Responsible for type checking and inference
- Detects many kinds of bugs
 - null pointer exceptions, array index out of bounds, ...
- Web: <https://checkerframework.org/>

Homework



- Assignment
 - <http://d3s.mff.cuni.cz/files/teaching/nswi154/ukoly/>
- Deadline
 - 1.12.2020 / 2.12.2020

Monitoring: Runtime Behavior & Software Development Process

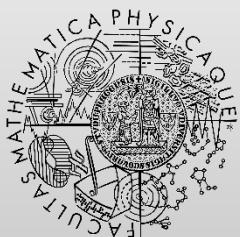
<http://d3s.mff.cuni.cz>

Department of
Distributed and
Dependable
Systems



Pavel Parízek

parizek@d3s.mff.cuni.cz



FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

Monitoring runtime behavior



Monitoring runtime behavior



- Goals
 - Recording information about program behavior
 - Notification about specific important events
- Information: performance, security, exceptions
- Target domain: long-running programs
 - Application servers (JBoss, Tomcat, WebSphere, ...)
 - Network servers and daemons (Apache, Sendmail)
- Alternative name: **tracing**

Basic approaches



- Manual implementation of logging commands
- Using tools for automated runtime monitoring

Tools



- Unix-like platforms
 - *Syslog, strace, ltrace, DTrace*
- Java ecosystem
 - *Log4j 2, Java Logging API, JVisualVM, JVM TI*
- Windows/.NET
 - *Log4net, NLog, Process Explorer*
- Events: custom messages, system calls, library calls
- Output: text log files (off-line inspection), GUI

Log4j



- Popular logging framework for Java platform
 - <http://logging.apache.org/log4j/2.x/>
- Features
 - Hierarchy of loggers based on class names
 - Filtering messages based on logging levels
 - Dynamically updateable configuration (XML)
 - Multiple output destinations (console, file)
 - Formatting log messages (printf-style, HTML)

Log4j API: example



```
import org.apache.logging.log4j.LogManager;  
import org.apache.logging.log4j.Logger;  
  
// get a Logger object with a particular name  
Logger logger = LogManager.getLogger("cz.cuni.mff");  
  
logger.warn("Running out of disk space");  
...  
logger.error("File {} not found", f.getName());  
...  
logger.info("Something normal happened");
```

Using Log4j



- Levels
 - TRACE < DEBUG < INFO < WARN < ERROR < FATAL
- Logger objects
 - Identified by logical names (e.g., Java class names)
 - They make a hierarchy based on the name prefixes
 - Logger named “cz.cuni” is a parent for the Logger “cz.cuni.mff”
 - Inheriting configuration (levels, appenders, formatting pattern)
 - Root Logger always exists at the top of any custom hierarchy
- Configuration: **XML**, programmatic
 - Default file name `log4j2.xml` (must be on classpath)

Configuration: example



```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration>
    <Appenders>
        <Console name="konzole" target="SYSTEM_OUT">
            <PatternLayout pattern="%d{HH:mm:ss} %-5level %c{36} - %m%n"/>
        </Console>
        <File name="logfile" fileName="test.log">
            <PatternLayout pattern="%d{HH:mm:ss} %-5level %c{36} - %m%n"/>
        </File>
    </Appenders>
    <Loggers>
        <Logger name="cz.cuni.mff" level="info">
            <AppenderRef ref="konzole"/>
        </Logger>
        <Root level="error">
            <AppenderRef ref="logfile"/>
        </Root>
    </Loggers>
</Configuration>
```

Appenders



- Responsible for writing log messages to actual target destinations
- Supported targets
 - Console (stdout, stderr)
 - File (buffered, appending)
 - Database (via JDBC)
 - SMTP (sending emails)
 - Network socket (TCP, UDP)
 - Unix/Linux syslog service

Layout



- Purpose: formatting messages
- Available layouts
 - Pattern
 - %m // message text
 - %n // line separator
 - %-5level // level, justified to the right, width five chars
 - %d{HH:mm:ss} // current datetime with pattern
 - %c{20} // logger name with the maximal length
 - %C %M %L // class name, method name, line number
 - %t // thread name
 - HTML, XML, Syslog

Tracing control flow



```
public Object doSomething(int arg1) {  
    logger.entry(arg1);  
    try {  
        ...  
        Object res = ...  
    }  
    catch (Exception ex) {  
        logger.catching(ex)  
    }  
    logger.exit(res);  
}
```

Log4j: other features



- Filtering messages
 - markers, regular expression, time
- Automatic reconfiguration
 - if you update the XML configuration file at runtime

Modern logging frameworks



- Simple Logging Facade for Java (SLF4J)
 - General unified API for logging frameworks
 - Supported backends: Log4j, logback, ...
 - <http://www.slf4j.org/>
- Logback
 - Replacement for Log4j
 - Implements SLF4J API
 - <http://logback.qos.ch/>

Logging platforms for .NET (C#, VB)



- Log4net
 - <http://logging.apache.org/log4net/index.html>
- NLog
 - <http://nlog-project.org/>
 - <https://github.com/NLog/NLog/wiki>
- Features
 - Configuration: file (XML), programmatic (API)
 - Multiple targets (file, database, console, email)
 - Layouts (plain text, CSV, XML, JSON)

Task 1

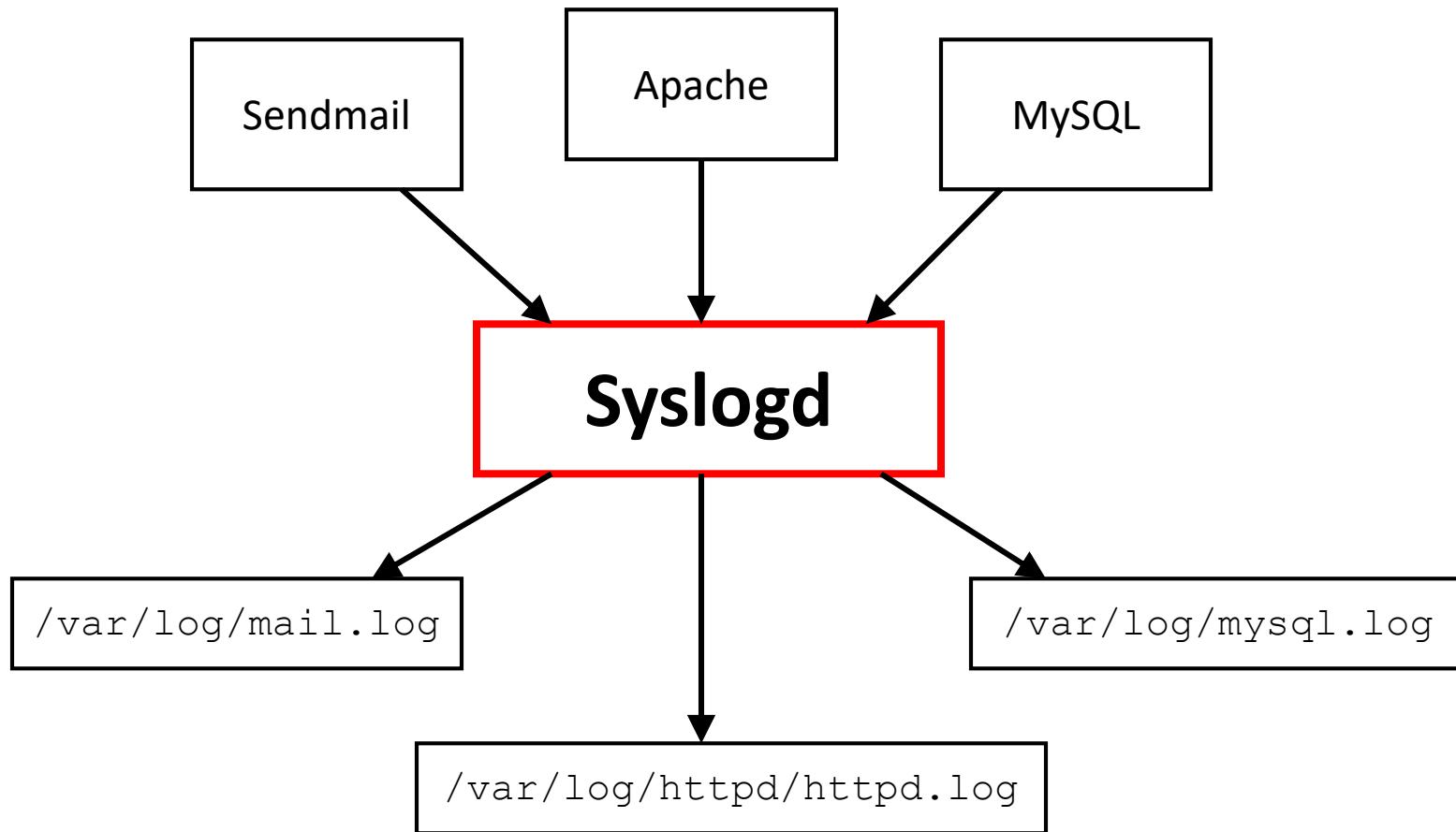
- Download Log4j/Log4net from the web
 - <http://logging.apache.org/log4j/2.x/>
 - Only important JAR files: core, api
 - <http://logging.apache.org/log4net/>
- Write simple program in Java or C#
 - You can also take some existing program (anywhere)
- Try important features of the particular logging framework
 - Use several Loggers
 - Different log levels
 - Configuration (XML)
 - Tracing control flow
- Check the output (console, log files)

Syslog



- Standard logging framework for Unix-like systems
- Service
 - Collecting messages from different sources (applications)
 - Writing received messages to various output destinations
 - log files (`/var/log`), another computer over network
 - Configuration: `/etc/syslog.conf`, `/etc/rsyslog.conf`
 - Log rotation: `/var/log/messages`, `/var/log/messages.1`, ...
- Protocol
 - Format of data exchanged between applications and the service
 - Message: content (plaintext, < 1024 bytes), priority
 - Supported priorities (low to high)
 - debug, info, notice, warning, error, critical, alert, emergency
 - Definition: RFC 3164, 3195

Configuration: example



Syslog API: example



```
#include <syslog.h>

openlog("myprog", LOG_CONS | LOG_PID, LOG_USER);

syslog(LOG_NOTICE, "Program runs for %d hours", 2);
syslog(LOG_ERROR, "File %s does not exist", fname);

closelog();
```

strace



- Tool for monitoring interactions with the operating system kernel
 - System calls performed by the given program
 - Signals received by the given program from OS
- Available for Unix-like platforms
- Usage: strace <program>
 - Attaching to a running process: strace -p <pid>
- Output: list of system calls and signals

```
open("/etc/passwd", O_RDONLY) = 3
```

```
open("/etc/passwords", O_RDONLY) = -1 ENOENT (No such file)
```



Task 2

- Try using
 - strace (syscalls)
 - ltrace (libraries)
- Check output

JConsole & JVisualVM



- Available in Oracle JDK
- Key features
 - Provides useful information
 - CPU usage, memory consumption, threads
 - Nice graphical interface
 - Connection to remote JVM
- How to run it: `jconsole` / `jvisualvm`
- Live demo
 - <http://d3s.mff.cuni.cz/files/teaching/nswi154/jpf-elevator.tgz>

Monitoring tools for C#/.NET



- .NET Memory Profiler
 - [https://marketplace.visualstudio.com/items?item
Name=SciTechSoftware.NETMemoryProfiler](https://marketplace.visualstudio.com/items?itemName=SciTechSoftware.NETMemoryProfiler)
- dotMemory
 - <https://www.jetbrains.com/dotmemory/>

Windows Sysinternals



- Process Explorer
 - <https://docs.microsoft.com/en-us/sysinternals/downloads/process-explorer>
 - Displays information about running processes
- Process Monitor
 - <https://docs.microsoft.com/en-us/sysinternals/downloads/procmon>
 - Displays some live (real-time) process activity

Other monitoring tools



- Instrumentation (binary, source code)
- Notification about specific events
 - accesses to object fields and variables
 - locking (acquisition, release, attempts)
 - procedure calls (e.g., user-defined list)
- Pin: dynamic binary instrumentation tool
 - <https://software.intel.com/en-us/articles/pin-a-dynamic-binary-instrumentation-tool>
- JVM Tool Interface (TI)
 - <https://docs.oracle.com/javase/8/docs/platform/jvmti/jvmti.html>
- Valgrind: heavyweight dynamic binary translation
- DiSL (<https://disl.ow2.org/bin/view/Main/>)

Log analysis tools



- Elasticsearch + Logstash + Kibana (ELK stack)
 - <https://www.elastic.co/>
- LOGalyze
 - <http://www.logalyze.com/>
- Splunk
 - <https://www.splunk.com/>
- Azure Monitor (Application Insights)
 - <https://azure.microsoft.com/en-us/services/monitor/>
- Prometheus
 - <https://prometheus.io/>

Monitoring development process



Issue tracking systems



- Typically part of a project management system
 - <https://github.com/>, <https://bitbucket.org/>
- Popular systems
 - Bugzilla, Trac, JIRA, YouTrack
- Issue = reported bug, feature request, other task
- Components
 - Some database of known issues
 - User interface (WWW, desktop)

Issue characteristics



- Time of reporting
- Product (module)
- Version of the product
- **Severity of the bug / Priority of the feature**
 - blocker, critical, major, normal, minor, enhancement
- Platform (OS, HW, SW)
- Textual comments
- **Current status**
 - new, unconfirmed, assigned, fixed, wontfix, resolved
- Assigned to
 - Who should do it (fix the bug, implement the feature)

Lifecycle of an issue (bug)

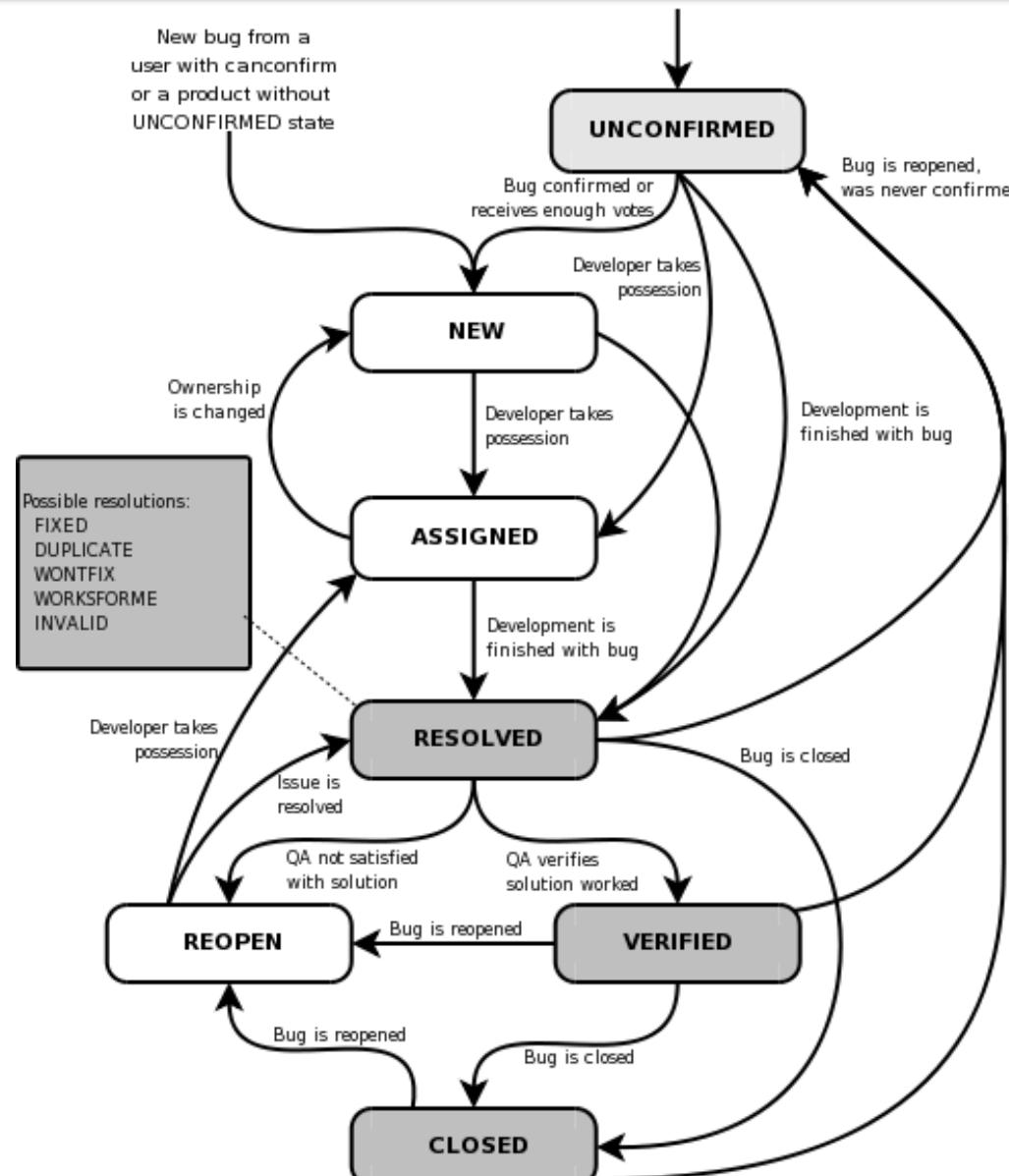


Figure taken from
<http://www.bugzilla.org/docs>



Common actions



- Developer
 - Entering new issues (bug reports)
 - Search for assigned tickets (issues)
 - Changing status of a specific ticket
- Manager
 - Inspecting overall statistics
 - Look for unresolved bugs
 - Assign priorities to features

Bugzilla



- Web-based tool
 - <http://www.bugzilla.org>
- SW requirements
 - Database (MySQL, PostgreSQL)
 - Perl 5 with specific modules
 - Web server (e.g., Apache httpd)
- Features
 - Advanced queries
 - Boolean operators (and, or, not)
 - Saved search
 - Cloning of bugs



- Project management system
 - <http://trac.edgewall.org/>
- Features
 - Tracking issues (bugs, feature requests)
 - Good integration with version control
 - Supported tools: Subversion, Mercurial, Git
 - Links from bug reports to source code files
 - Source code browser (version control)
 - Wiki pages (e.g., for documentation)

Test coverage



- Criteria: statement, branch, path
- Mutation testing
 - Detects missing tests
- Fault injection
- Practice: achieving 100% coverage is hard

Test coverage – tools



- Mutation testing and fault injection
 - Jester (<http://jester.sourceforge.net>)
 - Jumble (<http://jumble.sourceforge.net/>)
 - PIT (<http://pitest.org/>)
 - Major (<http://mutation-testing.org/>)
 - Nester (<http://nester.sourceforge.net/>)
 - NinjaTurtles (<http://www.mutation-testing.net/>)
- Coverage analysis
 - Cobertura (<http://cobertura.sourceforge.net/>)
 - Clover (<http://www.atlassian.com/software/clover/>)
 - dotCover (<https://www.jetbrains.com/dotcover/>)
 - JaCoCo (<https://www.eclemma.org/jacoco/>)
 - Support in Visual Studio (Test Explorer)

Continuous integration



- Frequent merge, building, and test execution
 - https://en.wikipedia.org/wiki/Continuous_integration
- Jenkins (<https://jenkins.io/>)
- Cruise Control (<http://cruisecontrol.sourceforge.net/>)
- TeamCity (<http://www.jetbrains.com/teamcity/>)
- Travis CI (<https://travis-ci.org/>)
- AppVeyor (<https://www.appveyor.com/>)

Other links



- Syslog
 - http://www.gnu.org/software/libc/manual/html_node/Syslog.html
- DTrace
 - <http://dtrace.org/blogs/about/>
- JConsole
 - <http://docs.oracle.com/javase/8/docs/technotes/guides/management/jconsole.html>
- Swiss Java Knife
 - <https://github.com/aragozin/jvm-tools>
- YouTrack
 - <https://www.jetbrains.com/youtrack/>
- JIRA
 - <https://www.atlassian.com/software/jira>

Homework



- Assignment
 - <http://d3s.mff.cuni.cz/files/teaching/nswi154/ukoly/>
- Deadline
 - 8.12.2020 / 9.12.2020

Auto-Generating Documentation & Source Code

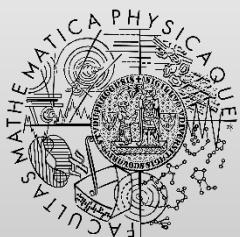
<http://d3s.mff.cuni.cz>

Department of
Distributed and
Dependable
Systems



Pavel Parízek

parizek@d3s.mff.cuni.cz



FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

Documentation



Types



- Developer
 - System architecture (design)
 - Code documentation
 - **API (methods, interfaces)**
 - Internally used algorithms
- User
 - Tutorials, guides, and examples
 - System administration manual

Documentation generators



- Main features
 - Extracting annotations from source code comments
 - Various output formats (HTML, PDF, LaTeX, man)
 - Generating navigation (links, references, indexes)
- Tools
 - Input: documentation written by the user as source code annotations (comments)
 - **Doxxygen**, JavaDoc, NDoc, Sandcastle

Doxygen



- Supported platforms: Unix/Linux, Windows
- Languages: C/C++, Java, C#, PHP, Python, etc
- Annotations format: JavaDoc style, Qt-style
- Output formats: HTML, PDF, LaTeX, man pages
- Released under GPL (open source)
- Home page
 - <http://www.doxygen.nl/>

How to run Doxygen



- Creating the default configuration file

```
doxygen -g Doxyfile
```

- Generating documentation from annotations

```
doxygen Doxyfile
```

Configuration



PROJECT_NAME =

INPUT = <dir>

PROJECT_NUMBER =

RECURSIVE = YES

PROJECT_BRIEF =

FILE_PATTERNS =

OUTPUT_DIRECTORY = <dir>

GENERATE_LATEX = NO

GENERATE_TREEVIEW = YES

EXTRACT_ALL = YES

EXTRACT_PRIVATE = YES

JAVADOC_AUTOBRIEF = YES

EXTRACT_STATIC = YES

QT_AUTOBRIEF = YES

Source code annotations: JavaDoc style



```
/**  
 * Returns the index of the first occurrence of the  
 * specified substring, starting at the given index.  
 * If the specified substring is not found, then -1  
 * is returned. The method can also throw exception.  
  
 * @param str the substring for which to search  
 * @param fromIndex the index from which to start  
 * the search  
 * @return the index of the first occurrence of  
 * given substring, or -1  
 * @throws NullPointerException if the given  
 * substring is null  
 */  
public int indexOf(String str, int fromIndex) {  
    ...  
}
```

Department of



Based on the official API documentation for the `java.lang.String` class

Source code annotations: other styles



- Qt style

```
/*!
 * Returns the index of the first occurrence of the
 * specified substring, starting at the given index.
 * If the specified substring is not found, then -1
 * is returned.
 * \param str the substring for which to search
 * \param fromIndex the index from which to start
 * \return the index of the first occurrence of given
 *         substring, or -1
 * \throws NullPointerException if the string is null
 */
```

- C++ style

```
/// ...
/// ...
/// ...
```

Annotations



- Classes

```
/**  
 * Brief description (first sentence).  
 * Full details (the rest).  
 */  
public class MyData {
```

- Fields

```
/** description */  
private int someNumber;
```

Annotations



- Methods

```
/**  
 * Brief description of the method (first sentence).  
 * Full details (all text up to the first command).  
 * @param id description  
 * @param [out] data my output argument  
 * @tparam T template parameter  
 * @return error code  
 * @throws NullPointerException if some arg is null.  
 */  
public int compute(int id, char* data, T typ) {  
    ...  
    return 0;  
}
```

Task 1



- Try out basic usage of Doxygen
 - Look into the configuration file (`Doxyfile`) to see additional options
 - Try different settings of configuration variables
 - Write documenting annotations for some program
 - Some example program from the previous lectures or your own program (any supported language)
 - Check the generated output (HTML)

References



- Links to other classes

- Links to functions

- `function_name()`
- `function_name(<argument list>)`
- `class_name#function_name`
- Example

```
/** Use the method createInput() to prepare data. */
public void myProc1(Data arg) {
```

- *See also* links

```
/**
 * This procedure evaluates the input expression.
 * @sa createInputExpr
 */
void process(Expr e) {
```

Where to put annotations



- Right before the corresponding declaration

```
/**  
 * ...  
 */  
class MyData {
```

- Almost anywhere if you specify the name

- file MyData.java

```
class MyData { ... }
```

- some other file

```
/**  
 * @class MyData  
 * ...  
 */
```

Annotating other entities



- Source code files

```
/**  
 * @file mydefs.h  
 * ...  
 */
```

- Packages (Java)

```
/**  
 * @package cz.cuni.mff  
 */
```

- Namespaces (C++, C#)

```
/**  
 * @namespace gui  
 */
```

Formatting



- HTML commands
 - Structure: <h1>, <h2>,
, <p>
 - Lists: , ,
 - Font: , <i>, <code>, <small>
 - Tables: <table>, <td>, <tr>, <th>
- Custom stylesheet (CSS)

Index page



```
/**  
 * @mainpage Program  
 * @section intro Introduction  
 * some text and HTML  
 * @section impl Implementation  
 */
```

Doxygen: advanced topics



- Grouping annotations (modules)
- Markdown syntax (formatting)
- Mathematical formulas (LaTeX)
- Visualizing relations between code elements
 - Example: inheritance diagrams, call graphs
 - Rendering: Graphviz (the “dot” tool)
- Customizable output
 - layout, colors, navigation
- Linking external documents

Task 2



- Try advanced features of Doxygen
 - Links and references
 - Annotating files
 - Formatting output
 - Main page (index)

JavaDoc



- Part of the standard Java platform
- Input for the generator
 - Java source code files with annotations
 - Comment files (package, overview)
- Output formats: HTML
- Annotation must precede the code element

JavaDoc: features



- Good support for inheritance (method overriding)
 - Copying parts of annotations from superclasses
 - Linking to superclasses and interfaces
- Documenting packages
 - Option 1: package-info.java

```
/**  
 * ...  
 */  
package cz.cuni.mff;
```
 - Option 2: package.html
 - File saved into the same directory as the .java source files

Running JavaDoc



- Command line

- javadoc -d myapp/doc -private
-sourcepath ./projects/myapp/src
cz.cuni.myapp cz.cuni.myapp.util
-subpackages cz.cuni.myapp.core

- Ant task

```
<javadoc destdir=". / doc">
    <packageset dir="${src.dir}">
        <include name="cz/cuni/myapp"/>
        <include name="cz/cuni/myapp/util"/>
        <include name="cz/cuni/myapp/core/**"/>
    </packageset>
</javadoc>
```

Customizing JavaDoc output



- Doclet
 - Extract some information about input Java classes
 - Print all the information in a custom format (style)
- Taglet
 - Define custom tag that can be used in annotations
 - Generates the output of a custom tag (formatting)

Code indexing



- Purpose
 - easy navigation, code browsing and searching
- Tools
 - Ctags
 - Generates large index of names in the source code
 - Integration with many editors (Vim, Emacs, jEdit)
 - Backend for many other tools (mostly Unix/Linux)
 - Supports many languages: C/C++, Java, C#, PHP, TeX
 - <http://ctags.sourceforge.net/>
 - OpenGrok
 - <https://oracle.github.io/opengrok/>

OpenGrok



- Toolset for indexing and presenting large source code repositories (Linux, NetBSD)
 - Based on Ctags
- Output
 - Set of inter-linked HTML files derived from sources
- Example
 - <https://nxr.netbsd.org/>

Code browsing tools



- Front-end
 - <https://cs.opensource.google/>
 - <https://developers.google.com/code-search/reference>
- Key features
 - Powerful search language, history, blame
 - Cross-referencing via graphs
 - linking declarations to usage
- Back-end
 - <https://kythe.io/>
 - <https://github.com/TreeTide/underhood>
 - <https://about.sourcegraph.com/>

Code Generation



Code Generation



- Writing code manually
 - Hard, tedious, and time-consuming work
 - Very error prone (copy & paste mistakes)
- Automated generating (partially)
 - From simple and high-level description
 - Input: template, database, model, UML

Options



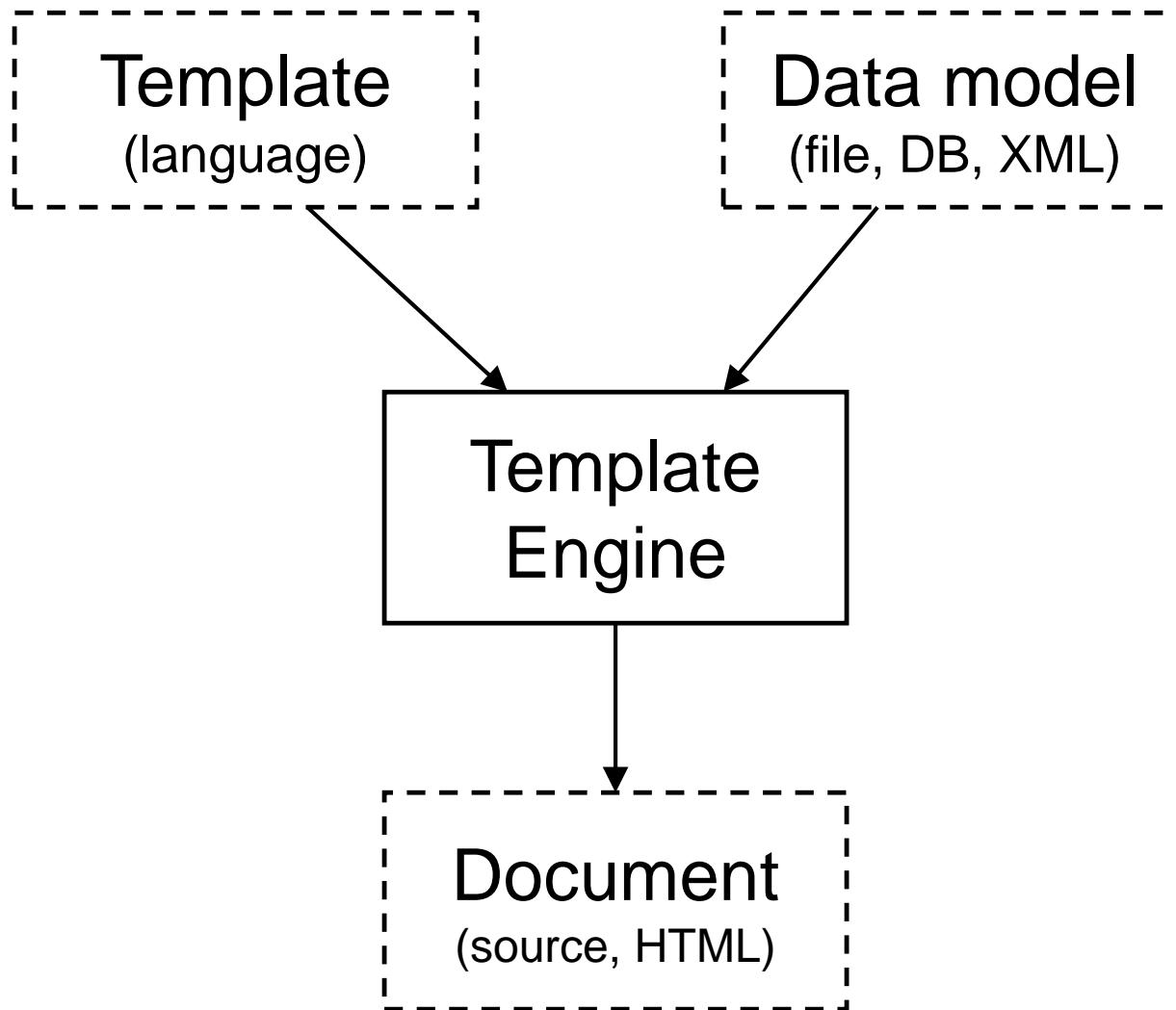
- Wizards (Eclipse, NetBeans, Visual Studio)
- Code skeletons from design models (UML)
- Parser generators (ANTLR, JavaCC, Bison)
- Generating code with **template engines**

Template engines



- General programming
- Domain specific (Web)
- Tools (frameworks)
 - FreeMarker, T4, StringTemplate, AutoGen

Using template engines



FreeMarker



- General-purpose template engine
 - Open source (BSD license)
 - <https://freemarker.apache.org/>
- Target platform: **Java**
 - Easily embeddable in Java programs
 - generic programs, Servlet and JSP containers
 - Special support for web development
 - Generating HTML pages from your templates

How to use FreeMarker



- Input
 - Template
 - Defined in the FreeMarker template language (FTL)
 - Data model
 - Prepared in the Java program
- Running
 - Template processor executed also in Java

FTL: example

```
<table>
<tr><th>Name</th><th>Salary</th></tr>
<#list employees as emp>
    <tr>
        <td>${emp.name}</td>
        <td>
            <!-- print top salaries in bold -->
            <#if (emp.salary > 2000)><b>${emp.salary * 2}</b>
            <#else>${emp.salary + 500}
            </#if>
        </td>
    </tr>
</#list>
</table>
```



FTL: other features



- Direct access to sequence elements
 - `${employees[2].salary}`
- Custom procedures and functions
 - First-class language constructs (assignable)
 - Invoking custom function: `${add(2, 3)}`
- Including other files
 - `<#include "header.html">`
- Custom directives
 - `<@mytag> ... </@mytag>`

Data model: example



```
(root)
|
| -- employees
|     | -- [1]
|         | -- name = "Joe Doe"
|         | -- salary = 1800
|     |
|     | -- [2]
|         | -- name = "John Smith"
|         | -- salary = 2500
|
| -- products
...
...
```

Preparing the data model



```
Map data = new HashMap();
List employees = new LinkedList();
Map emp = new HashMap();
emp.put("name", "Joe Doe");
emp.put("salary", new Integer(1800));
employees.add(emp);
...
data.put("employees", employees);
```

Executing template processor



- Initialization of FreeMarker
- Loading template from file
- Preparing the data model
- Applying template on data

Initialization



```
Configuration cfg = new Configuration();  
  
cfg.setDirectoryForTemplateLoading(  
    new File("resources/templates")  
) ;  
  
cfg.setObjectWrapper(new DefaultObjectWrapper());  
  
cfg.setDefaultEncoding("UTF-8");  
cfg.setTemplateExceptionHandler(  
    TemplateExceptionHandler.RETHROW_HANDLER  
) ;  
cfg.setIncompatibleImprovements(  
    new Version(2,3,20)  
) ;
```

Processing template



- Loading

- ```
Template tl =
cfg.getTemplate("test.ftl");
```

- Applying

- ```
FileWriter out =  
new FileWriter("index.html");
```
 - ```
tl.process(data, out);
```
  - ```
out.flush();
```

How to define custom functions



```
public class AddMethod implements TemplateMethodModel {  
    public TemplateModel exec(List args) {  
        Integer op1 = new Integer((String) args.get(0));  
        Integer op2 = new Integer((String) args.get(1));  
        return new SimpleNumber(new Integer(op1 + op2));  
    }  
}  
  
data.put("add", new AddMethod());
```

Task 3

- Download FreeMarker
 - <http://sourceforge.net/projects/freemarker/files/freemarker/2.3.20/>
 - <https://freemarker.apache.org/freemarkerdownload.html>
- Write template, data model, and processing code
 - Option 1: Generating classes from a list of field names and types (declarations, getters and setters, equals)
 - Option 2: Generating code that will create GUI just from a simple textual description (widgets, labels, positions)
 - Option 3: your own idea (e.g., something that you need)
- Specify data model in the Java program
- Use arbitrary output language (C#, C, Java, ...)



- Text Template Transformation Toolkit
- Target platform: C#, VB (.NET)
- Support: Visual Studio, MonoDevelop
- Web: <https://docs.microsoft.com/en-us/visualstudio/modeling/code-generation-and-t4-text-templates?view=vs-2015>

Links



- Doxygen
 - <http://www.doxygen.nl/>
- JavaDoc
 - <http://docs.oracle.com/javase/8/docs/technotes/tools/unix/javadoc.html>
 - <http://docs.oracle.com/javase/8/docs/technotes/guides/javadoc/index.html>
- NDoc
 - Platform C#/.NET, documentation comments written in XML
 - <http://ndoc.sourceforge.net/>
 - <https://sourceforge.net/p/ndoc3/wiki/Home/>
- Sandcastle
 - Help file builder for Windows/.NET
 - <https://github.com/EWSSoftware/SHFB>
 - <http://sandcastle.codeplex.com/>
- Further information (recommended)
 - <http://www.literateprogramming.com/documentation.pdf>

Links



- StringTemplate
 - <http://www.stringtemplate.org/>
- Project Lombok
 - <http://projectlombok.org/>
- GNU AutoGen
 - <http://www.gnu.org/software/autogen/>

Homework



- Assignment
 - <http://d3s.mff.cuni.cz/files/teaching/nswi154/ukoly>
- Deadline
 - 15.12.2020 / 16.12.2020

Performance Analysis

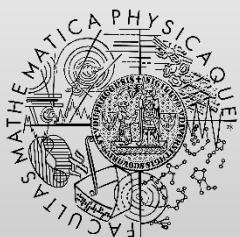
<http://d3s.mff.cuni.cz>

Department of
Distributed and
Dependable
Systems



Pavel Parízek

parizek@d3s.mff.cuni.cz



FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

Performance analysis



- Find where the program spends most time
 - Identify code that you should optimize for speed
- Call graph: function names and spent time
- Important performance characteristics
 - throughput, latency, maximal load, average request processing time, ...
- Main approaches
 - **profiling**, benchmarking, load testing

Profiling



- Tools measuring frequency and duration of procedure calls during program execution
 - **GProf**, OProfile, Valgrind
- Basic principles (how it works)
 - Sampling: results not precise for short time periods
 - Recording program counter (PC) at regular intervals
 - Program instrumented with profiling-related code
 - Getting information from HW performance counters



- GNU Profiler
 - Distributed as a part of binutils
- Documentation
 - <https://sourceware.org/binutils/docs/gprof/>

How to use GProf



1) Build program with enabled profiling

```
gcc -g -pg -o program program.c
```

- Instrumentation: code that collects raw timing data added to the entry and exit points of each function

2) Execute the program normally

- Raw profile data written to the file gmon.out

3) Generate statistics (tables with results)

```
gprof <options> program [gmon.out]
```

- Output: flat profile, call graph

Flat profile



- How to get it

- `gprof -p program [gmon.out]`
- Excluding specific function
 - `gprof -p -P<function_name> program`

%	cumulative	self		self	total	
time	seconds	seconds	calls	ms/call	ms/call	name
35.29	0.06	0.06	3	20.00	43.33	compute
32.35	0.12	0.06	14000896	0.00	0.00	S_n
17.65	0.14	0.03	3	10.00	53.33	get_msg
5.88	0.15	0.01	5000320	0.00	0.00	F
5.88	0.17	0.01				main

Demo 1



- Basic features of GProf
 - Generating the flat profile
 - Excluding some functions
- Subject program
 - <http://d3s.mff.cuni.cz/files/teaching/nswi154/sha.tgz>
- Program has to run for a long time (at least few seconds) to get useful results
 - Measurement results are invalid otherwise

Flat profile: source code lines



- How to get it
 - gprof **-p -l** program

Each sample counts as 0.01 seconds.

%	cumulative	self		
time	seconds	seconds		name
17.65	0.03	0.03	S_n (sha.c:18 @ 80485f9)
11.76	0.05	0.02	S_n (sha.c:17 @ 80485f0)
11.76	0.07	0.02	compute (sha.c:152 @ 804895b)
11.76	0.09	0.02	get_msg (sha.c:192 @ 8048baa)

Call graph



- How to get it
 - gprof **-q** program

		index	% time	self	children	called	name
<hr/>							
[2]	94.1	0.03	0.13	0.03	0.13	3/3	main [1]
		0.03	0.13	0.03	0.13	3	get_message_digest [2]
		0.06	0.07	0.06	0.07	3/3	compute_digest [3]
		0.00	0.00	0.00	0.00	3/3	get_padded_length [10]
		0.00	0.00	0.00	0.00	3/3	padd_message [11]
<hr/>							
[3]	76.5	0.06	0.07	0.06	0.07	3/3	get_message_digest [2]
		0.06	0.07	0.06	0.07	3	compute_digest [3]
		0.06	0.00	14000896/14000896	14000896/14000896	S_n [4]	
		0.01	0.01	5000320/5000320	5000320/5000320	F [5]	
<hr/>							

Demo 2



- GProf: other features
 - Flat profile for source code lines
 - Call graph (reading the output)

Performance analysis



- It is hard and tricky
 - Profiling results not 100% precise
 - Statistical approximation is used
 - Many things influence performance
 - Resource sharing (caches), garbage collection
 - Even harder for programs in JVM / .NET CLR
- Recommended practice
 - Use profilers only to identify parts of your program that are much slower than others



- GUI profiler for Java (heap, CPU)
- Documentation
 - <https://visualvm.github.io/>
 - <http://docs.oracle.com/javase/8/docs/technotes/guides/visualvm/index.html>
- Important features
 - Heap dump
 - CPU sampling

Visual Studio profiling



- Available tools
 - CPU usage, memory usage, and many others
- Web (documentation, tutorials)
 - <https://docs.microsoft.com/en-us/visualstudio/profiling/?view=vs-2019>

Other profiling tools



- YourKit
 - Powerful profiler for Java and .NET
 - <http://yourkit.com/home/index.jsp>
 - Many advanced features (see web)
 - Handles also very large applications
- dotTrace
 - Target platform: C#, .NET applications
 - <https://www.jetbrains.com/profiler/>
- Valgrind
 - Supported tools: Cachegrind, Callgrind, Massif, DHAT, ...
 - Running: --tool=<cachegrind | callgrind | massif>
 - Inspecting results: cg_annotate, callgrind_annotate, ms_print
 - Demo 3: using tools on some program

Load testing



- Generating specific (heavy) load for server applications (WWW, email, database)
 - Target URL and payload
 - Number of threads (clients)
 - Frequency of requests
- JMeter (<http://jmeter.apache.org/>)
 - supports: GUI, command-line, distributed mode
- Netling (<https://github.com/hallatore/Netling>)

Coverage



- Metrics
 - Statement coverage
 - Branch coverage
 - Control-flow paths
- Tools
 - GCov (<https://gcc.gnu.org/onlinedocs/gcc/Gcov.html>)
 - JCov (<https://wiki.openjdk.java.net/display/CodeTools/jcov>)
 - JaCoCo (<http://www.jacoco.org/jacoco/>)

Measuring coverage with Gcov



- Build program with special options
 - `gcc -fprofile-arcs -ftest-coverage -o program program.c`
- Execute the program normally
- Run the gcov tool on source code files
 - `gcov program.c`
- Open the file `program.c.gcov`
- With branch and block statistics
 - `gcov -b program.c`

Related courses



- NSWI131: Vyhodnocování výkonnosti počítačových systémů
 - Topics: benchmarking, experimental evaluation, statistical analysis, modeling, simulation
- NSWI126: Pokročilé nástroje pro vývoj a monitorování software
 - Topics: other profilers and performance analyzers
 - LS 2019/2020

Homework



- Assignment
 - <http://d3s.mff.cuni.cz/files/teaching/nswi154/ukoly/>
- Deadline
 - 31.12.2020
- Homework targets GProf and programs in C
 - Alternative: you can use another profiler (e.g., for Java or C#) on the program of your choice
 - Report answers to performance-related questions very similar to those specified in tasks 5a-d, 5f and 7a-g
 - For questions inspired by tasks 7a-g, pick some functions in the program code instead of 'zip' and 'updcrc'