

Laboratorio di Programmazione

Prof. Marco Bertini marco.bertini@unifi.it http://www.micc.unifi.it/bertini/



Code versioning: techniques and tools

Software versions

- All software has multiple versions:
 - Each time you edit a program
 - Versions within a development cycle
 - Test release with debugging code
 - Alpha, beta of final release
 - Variations for different platforms
 - Hardware and software
 - Different releases of a product

Version control

- Version control tracks multiple versions of code.
- In particular, allows:
 - old versions to be recovered
 - multiple versions to exist simultaneously
- Typically multiple users can contribute to software development and version control systems allow them to collaborate:
 - multiple versions of multiple users, merging their contribute
 - tracks who did what



 Version control tracks multiple versions of code In general version control (or revision control, or source control) is about managing multiple versions of documents, programs, web sites.
 It works best on text documents but can manage also binary files such as images. to collaborate:

- multiple versions of multiple users, merging their contribute
- tracks who did what

Why using version control?

- Because it is useful
 - You will want old/multiple versions
 - Without version control, can't recreate project history
 - Allows to go back in history, to solve bugs introduced since the last version of the code
- Because everyone does
 - A basic software development tool.
 Beware of those who do not use it.
 - If you need to share coding responsibilities or maintenance of a codebase with another person, you need version control.

Why using version control?

- For working by yourself:
 - Gives you a "time machine" for going back to earlier versions
 - Gives you great support for different versions of the same project
- For working with others:
 - Greatly simplifies concurrent work, merging changes



- A Code Base does not just mean code! It also includes:
 - Documentation
 - Build Tools (CMake files, Makefiles, etc.)
 - Configuration files
- All these files may change over time and older versions have to be kept.

Code base

Manage these things using a version control system (VCS)

A version control system is a system which allows for the management of a code base.

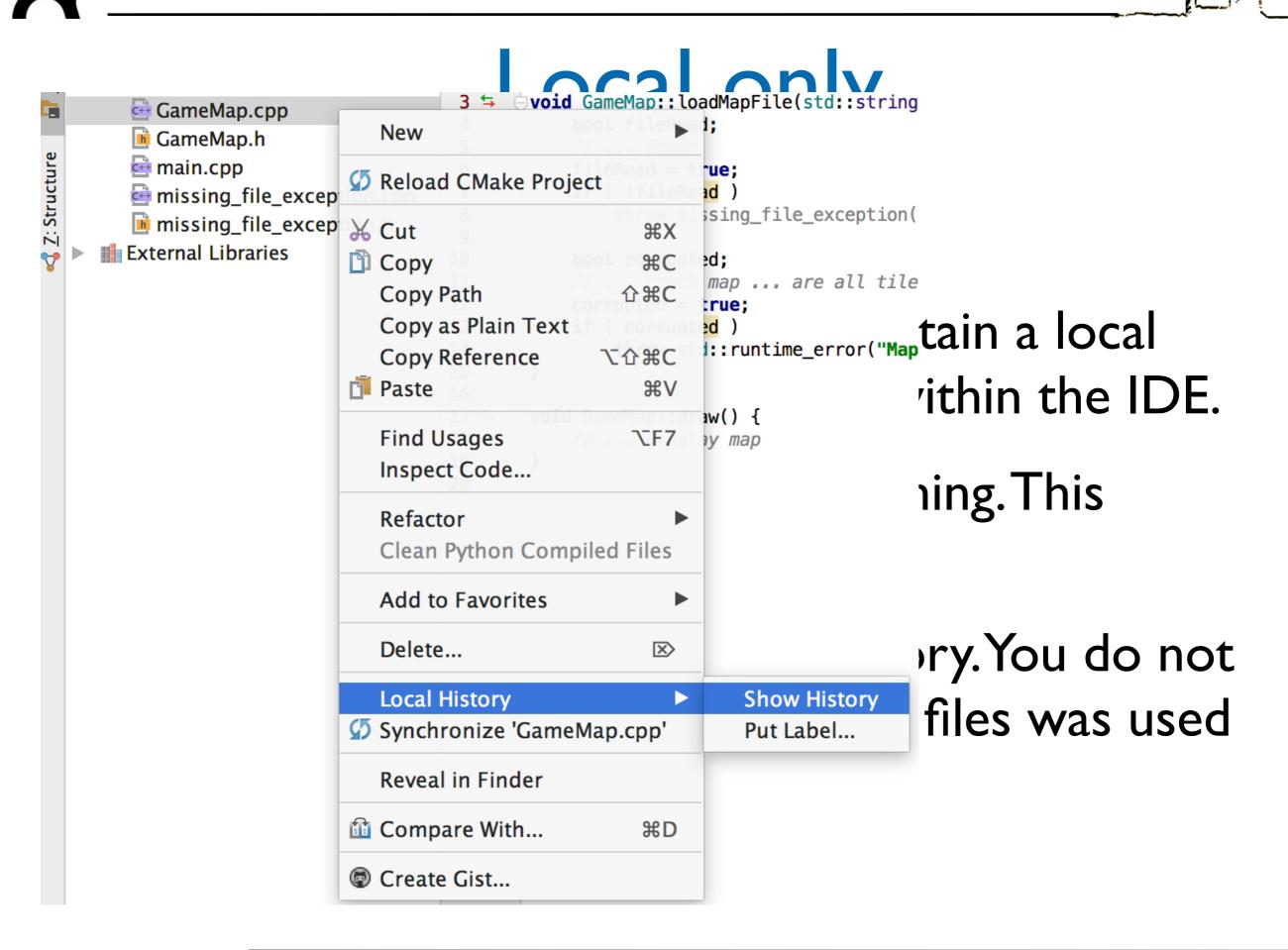
- Documentation
- Build Tools (CMake files, Makefiles, etc.)
- Configuration files
- All these files may change over time and older versions have to be kept.

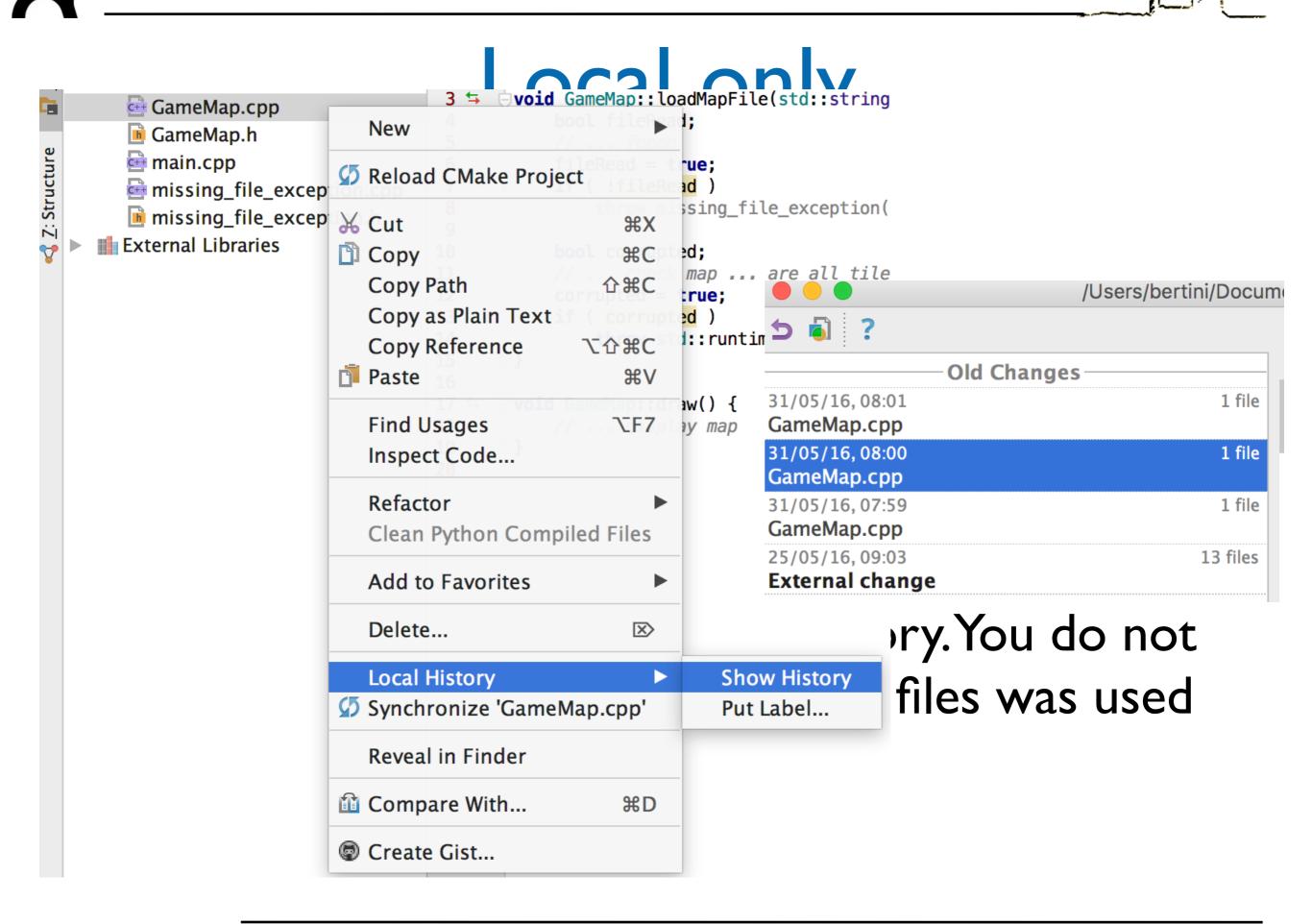
Types of Version Control Systems

- Local only keeps a local database of changes in your local machine filesystem.
- Centralized (Subversion, CVS), require a connection to a central server and "checkout"
- Distributed (Git, Mercurial) allow for local systems to be "mirrors" of the central repo. You don't need to be connected to the central server to get work or commits done.



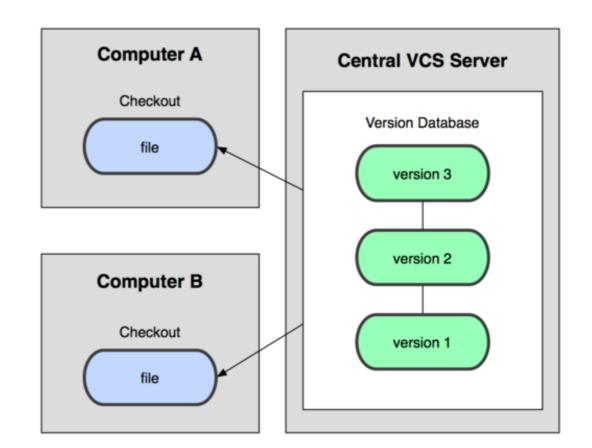
- IDE like CLion and Eclipse maintain a local history of each file developed within the IDE.
- Pros: you don't have to do anything. This versioning is automatic.
- Cons: each file has its own history. You do not know which versions of several files was used at a certain moment.





Centralized

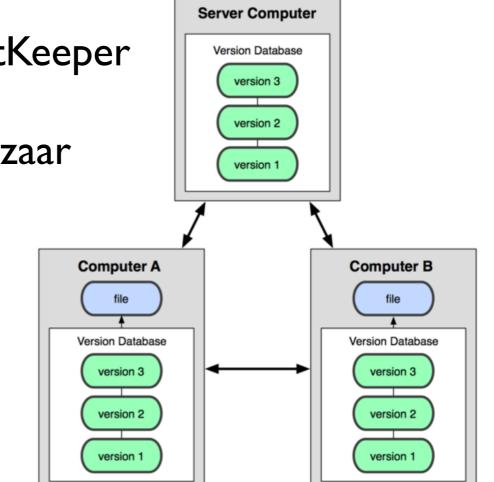
- Traditional version control system
 - Server with database
 - Clients have a working version
- Examples
 - CVS
 - Subversion
 - Visual Source Safe
- Challenges
 - Multi-developer conflicts
 - Client/server communication



Distributed

- Authoritative server by convention only
- Every working checkout is a repository
- Get version control even when detached
- Backups are trivial

- Examples
 - Git
 - Mercurial
 - BitKeeper
 - Bazaar



Overview of the process

- Files are kept in a **repository**
- Repositories can be local or remote to the user
- The user edits a copy called the **working copy**
- Changes are **committed** to the repository when the user is finished making changes
- Other people can then access the repository to get the new code
- Can also be used to manage files when working across multiple computers

Branching

- Branches allows multiple copies of the code base within a single repository.
- Different customers have different requirements
 - Customer A wants features A, B, C
 - Customer B wants features A & C but not B because his computer is old and it slows down too much.
 - Customer C wants only feature A due to costs
- Each customer has their own branch.
- Different versions can easily be maintained

Basic features of a VCS

- Check-in and check-out of items to repository
- Creation of baselines (labels/tags)
 - e.g. "Version 1.0 released!"
- Control and manipulation of branching
 - management of multiple versions
- Overview of version history
 - Allows to see who changed what

Check out / check in

- If you want to make a change the file needs to be checked out from the repository.
- When changes are completed the new code is **checked-in**.
- A **commit** consists of a set of checked in files and the diff between the new and parent versions of each file.
- Each check-in is accompanied by a user name and other meta data.
- Check-ins can be exported from the Version Control System the form of a **patch**.

Revision

- Consider
 - Check out a file
 - Edit it
 - Check the file back in
- This creates a new version of the file
- With each revision, system stores
 - The diffs for that version (typically for efficiency, the VCS doesn't store entire new file, but stores diff with previous version)
 - The new file version number
 - Other metadata
 - Author
 - Time of check in
 - Log file message

Merge

- There are occasions when multiple versions of a file need to be collapsed into a single version.
- E.g. a feature from one branch is required in another, or two developers worked on the same file.
- This process is known as a **merge**.





• There are occasions when multiple versions of

a file need to be collarsed into a single

	ixis Me	erge - [Z:WocsVAation TestWife2.txt	, Z:Woc	sVAati	ion TestNife3.txt, and Z:\Docs\Amation Te	est Wife	.txt]		
<u>a</u>	File	<u>E</u> dit <u>V</u> iew <u>W</u> indow <u>H</u> elp						_ = ×	
*P	1				🖸 🖻 🎒 🖆 🖗 🐴				
3		🗛 🕺 🖻 🖾 🗠 🗠	â ,	2	± *• •• */ /• *• •• */	-			
Z:\Do	cs\Ara:	xis Docs\Artwork\Experimental 🛄 💽 🚅		Z:\Docs	s\Araxis Docs\Artwork\Experimental 🛄 💽 📑		Z:\Docs	\Araxis Docs\Artwork\Experimental 🛄 🚽 🖪	
			•			••	[equired in d on the
	39.0	(Local Improbability Field		39	(Local Improbability Field 🛆		39	(Local Improbability Field	ninrea in
	40			40			40		
		In order to test this hypot		41	In order to test this hypot		41	In order to test this hypot	
	42	the LIF hypothesis were cor		42	the LIF hypothesis were cor		42	the LIF hypothesis were cor	
	43	substantial enough as to en		43	substantial enough as to en		43	substantial enough as to en	d on tho
	44	집 같은 것이 많은 것이 집 같은 것		44			44		
	45	We all have an intuitive un		45	We all have an intuitive un		45		
	46	do stunningly well'. We kno		46	do stunningly well'. We kno		46	do stunningly well'. We kno	
	47	Te		47			47	Te	
19912	48	It would seem that all life		48	It would seem that all life		48	It would seem that all life	
2	49	rarely sees plants sufferin	а г	49	rarely sees plants sufferin	[49	rarely sees plants sufferin	
2		not bright) will often bur		50	not desperately bright) # *		50	not bright) will often bur#	
1	51	accessible when the squirre		51	accessible when the squirre		51	accessible when the squirre	
		Intelligent people are part		52	Intelligent people are part		52	Intelligent people are part	
	53 : 54	LIFE, but by and large they are said to be DULL).		53	LIFE, but by and large they		53 54	LIFE, but by and large they	
	54	are said to be DULL).		54	are said to be DULL).			are said to be DULL).	
	55 56			55 56			55 56	Des southers at a line shi i work of the	
		Thur and the standard of the set			Humanities students often s	\mathbf{i}		By contrast, intelligent p # an exam raises the likeliho	
	57	Humanities students often s therefore intelligent. Quit		57 58	therefore intelligent. Quit		57 58		
	58 59	against the obvious consequ		58	against the obvious consequ	1		breakthroughs are likely to to this). Software engineer	
	60 ·	therefore greatly raise the		60	therefore greatly raise the		59 60	disaster such as a hard dis	
	60 61	the computer lab burning do		61	the computer lab burning do	l	61	disaster such as a hard dis	
	62	the computer ian burning do		62	the computer iab burning do		62	Humanities students often s	
2.	- 24	So far, much anecdotal evi*	1 /	63	1. Strong statements affect		<i>1</i> 63	therefore intelligent. Quit	
	64	Marmalade Rule [1]. Even so		64	2. The effect of such state		64	against the obvious consequ	
	65	internation (1). Iven so		65	3. LIFE seems to centre on		65	therefore greatly raise the	
		1. Strong statements affect		66	4. LIF effects increase wit		66	the computer lab burning do	
¥	67	2. The effect of such state		67	5. It is possible to take p		67		
*		3. LIFE seems to centre on		68			68	1. Strong statements affect	
Ŧ	20	A TTP - 22-12		20	6. Such precautions reduce			The shift and the state of the	
<		>		<			<	>	

Merging

- I. Start with a file, e.g. v. I.5
- 2. Bob makes changes A to v.1.5
- 3. Alice makes changes B to v.1.5
- 4. Assume Alice checks in first
- 5. Current revision is v.1.6 = apply(B, v.1.5)
- 6. Now Bob checks in
- 7. System notices that Bob checked out v.1.5, but current version is v.1.6
- 8. Bob has not made his changes in the current version!

- 9. The system complains
- 10. Bob is told to update his local copy of the code
- II. Bob does an update
- I2. This applies Alice's changes B to Bob's code
- I 3. Two possible outcomes of an update:
 - Success
 - Conflicts



- Assume that:
 - apply(A, apply(B, v. I.5) = apply(B, apply(A, v. I.5))
- Then then order of changes didn't matter
- Same result whether Bob or Alice checks in first
- The version control system is happy with this
- Bob can now check in his changes
 - Because apply(B, apply(A, v. I.6)) = apply(B, v. I.6)



- Assume
- $apply(A,apply(B,I.5) \neq apply(B,apply(A,I.6))$
- There is a conflict
 - The order of the changes matters
 - Version control will complain
- Arise when two programmers edit the same piece of code
 - One change overwrites another





- System cannot apply changes when there are conflicts:
 - Final result is not unique
 - Depends on order in which changes are applied
- Version control shows conflicts on update
- Conflicts must be **resolved by hand**

Conflicts

- Conflict detection is based on "nearness" of changes
 - Changes to the same line will conflict
 - Changes to different lines will likely not conflict
- Note: Lack of conflicts does not mean Alice's and Bob's changes work together





• Merging is syntactic

- Semantic errors may not create conflicts
 - But the code is still wrong
- You are lucky if the code doesn't compile
 - Worse if it does ...

Problem example

- The Linux kernel runs on different processors (ARM, x86, MIPS). These can require significant differences in low level parts of the code base
- Many different modules
- Old versions are required for legacy systems
- Because it is open source, any one can download and suggest changes.





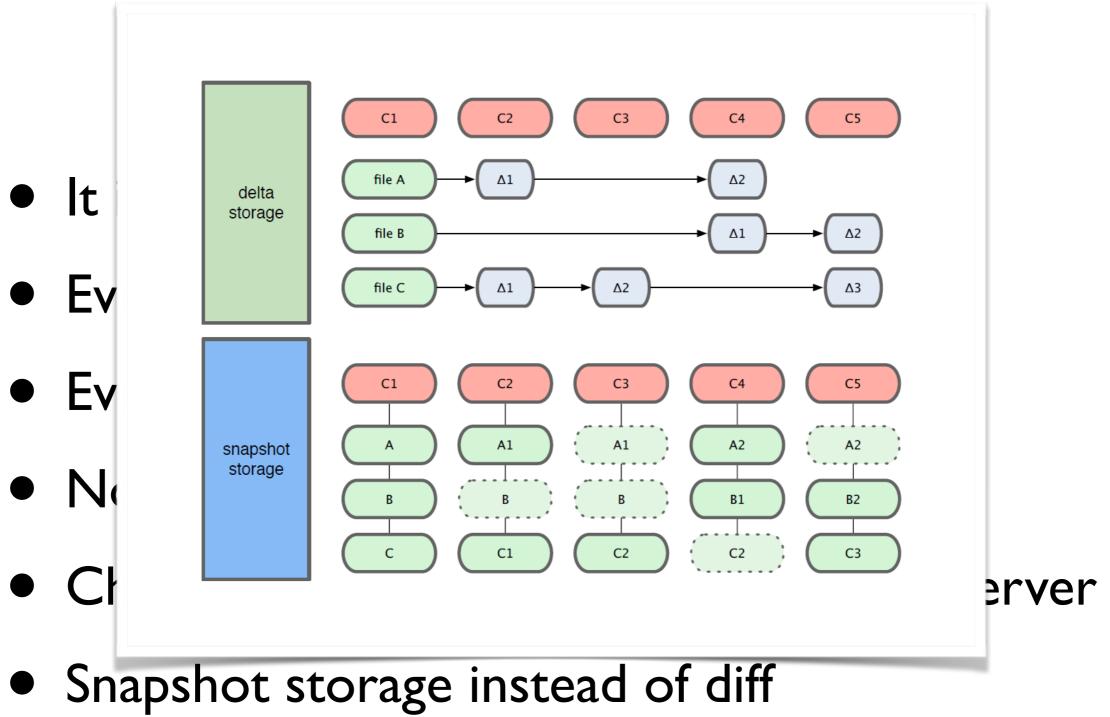
History

- Came out of Linux development community
- Linus Torvalds, 2005
- Initial goals:
 - Speed
 - Support for non-linear development (thousands of parallel branches)
 - Fully distributed
 - Able to handle large projects like Linux efficiently

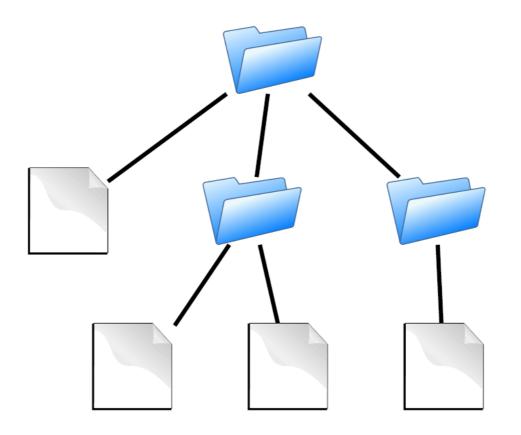
Features

- It is distributed
- Everyone has the complete history
- Everything is done offline
- No central authority
- Changes can be shared even without a server
- Snapshot storage instead of diff









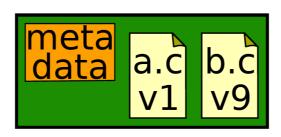
- Contains:
- directories
- files



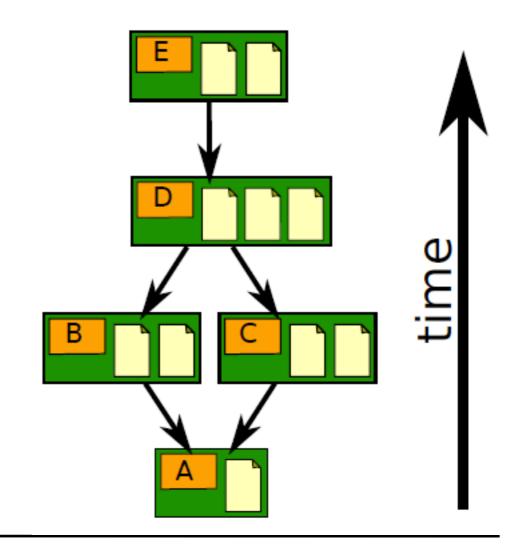


- files
- commits

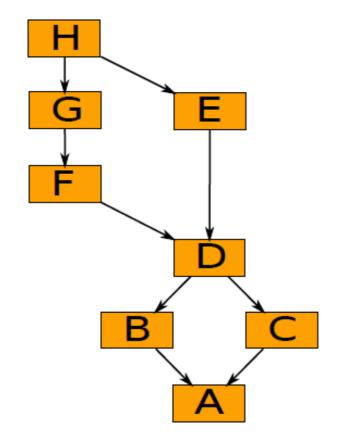




- ancestry relationships
- records history of changes

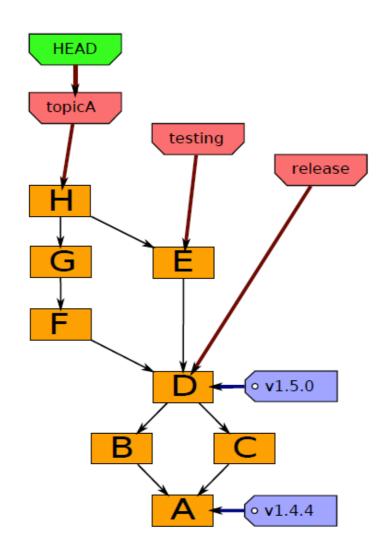


Ancestry relationships



 form a directed acyclic graph (DAG)

Ancestry graph features

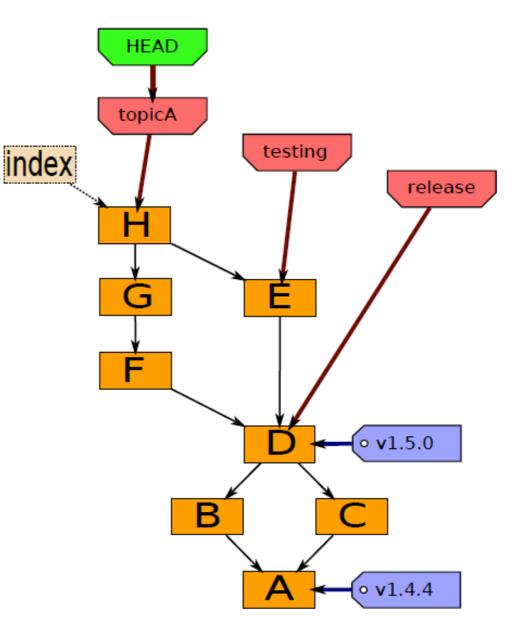




- is current checkout
- usually points to a branch



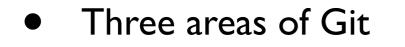
Git component



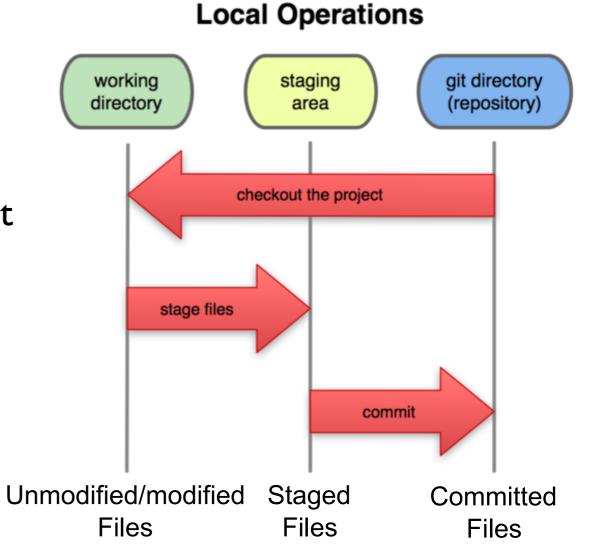
- Index
 - "staging area"
 - what is to be committed

Getting started

Index HEAD



- The HEAD
 - last commit snapshot, next parent
- Index
 - Proposed next commit snapshot
- Working directory
 - Sandbox





Basic workflow

- Init a repo(sitory): init to start a new project or clone an existing project
 - will create a ".git" directory. This is your local repo.
- Edit files
- Stage the changes (add files to repo)
- Review your changes
- Commit the changes

You can work as much as you like in your working directory, but the repository isn't updated until you commit something

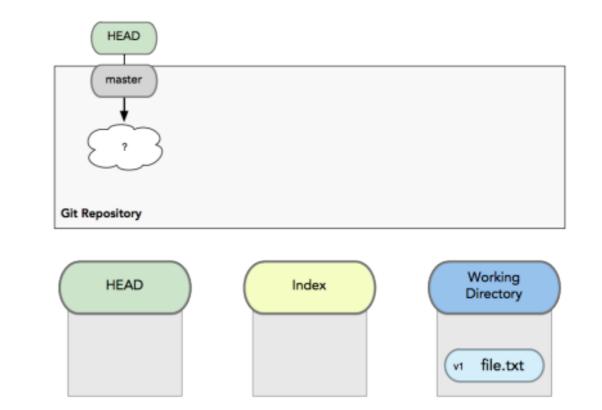
- Init a repo(sitory): init to start a new project or clone an existing project
 - will create a ".git" directory. This is your local repo.
- Edit files
- Stage the changes (add files to repo)
- Review your changes
- Commit the changes

What not to track

- It's important to tell Git what files you do not want to track
- Temp files, executable files, etc. do not need version control (and can cause major issues when merging!)
- We add the filenames to the special file .gitignore. We store this file in the repository

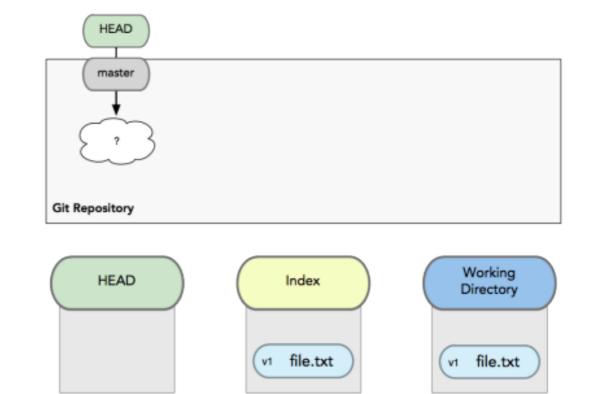
Getting started: edit file

- A basic workflow
 - Edit files
 - Stage the changes
 - Review your changes
 - Commit the changes





- A basic workflow
 - Edit files
 - Stage the changes
 - Review your changes
 - Commit the changes

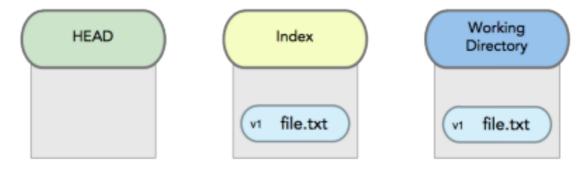


git add

Getting started: review

- A basic workflow
 - Edit files
 - Stage the changes
 - Review your changes





• Commit the changes

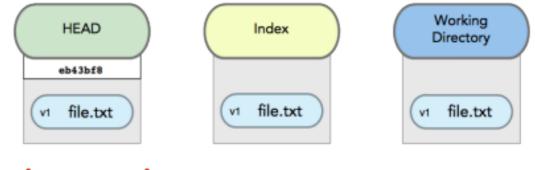
git status

zachary@zachary-desktop:~/code/gitdemo\$ git add hello.txt
zachary@zachary-desktop:~/code/gitdemo\$ git status
On branch master
Changes to be committed:
(use "git reset HEAD <file>..." to unstage)
#
modified: hello.txt
#

Getting started: commit

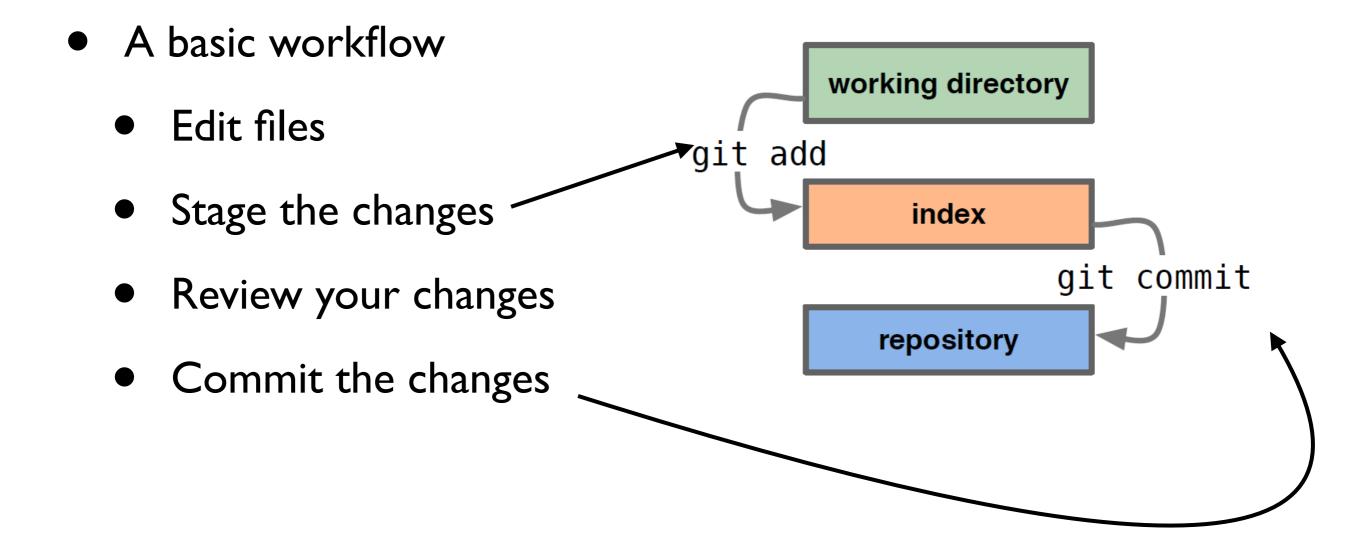
- A basic workflow
 - Edit files
 - Stage the changes
 - Review your changes
 - Commit the changes





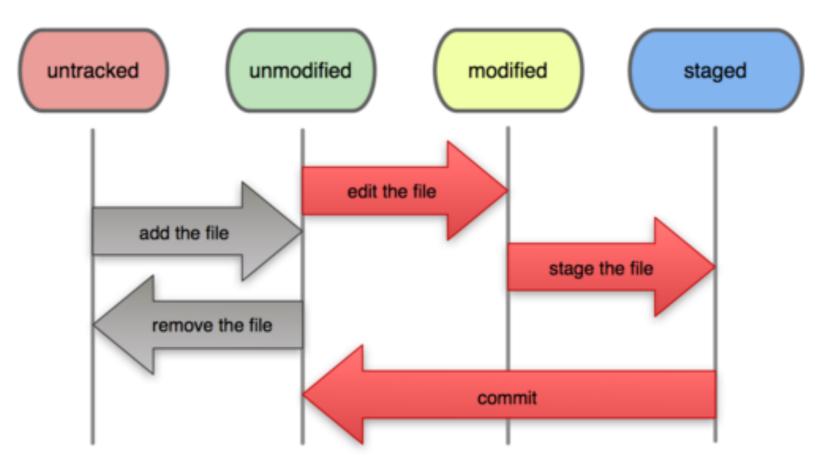
git commit





File life lifecycle

File Status Lifecycle



Files outside Git

Commits and graphs

- A commit is when you tell git that a change (or addition) you have made is ready to be included in the project
- When you commit your change to git, it creates a commit object, that represents the complete state of the project, including all the files in the project
- The very first commit object has no "parents"
- Usually, you take some commit object, make some changes, and create a new commit object; the original commit object is the parent of the new commit object
 - Hence, most commit objects have a single parent
 - You can also merge two commit objects to form a new one, in this case the new commit object has two parents
 - Hence, commit objects form a directed graph
- Git is all about using and manipulating this graph

Commits and graphs

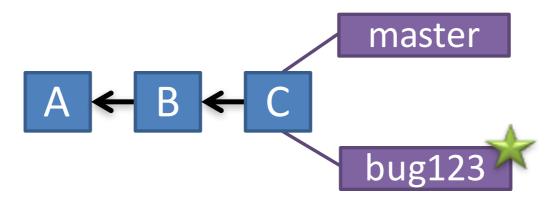
- A head is a reference to a commit object
- The "current head" is called HEAD (all caps)
- Usually, you will take HEAD (the current commit object), make some changes to it, and commit the changes, creating a new current commit object
- This results in a linear graph: $A \rightarrow B \rightarrow C \rightarrow ...$ \rightarrow HEAD

Good practice

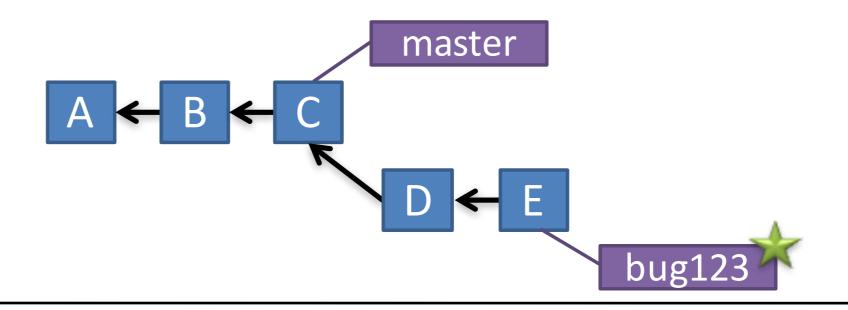
- In git, "Commits are cheap." Do them often.
- When you commit, you must provide a oneline message stating what you have done
 - Terrible message: "Fixed a bunch of things"
 - Better message: "Corrected the calculation of median scores"
- Commit messages can be very helpful, to yourself as well as to your team members

- Branch annotates which commit we are working on
- E.g. we can work on development, create a new branch to handle a bug, write code in the branch and then merge to the master branch

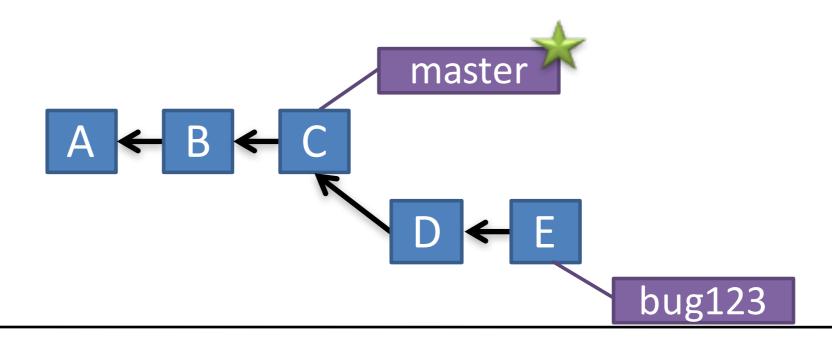
- Branch annotates which commit we are working on
- E.g. we can work on development, create a new branch to handle a bug, write code in the branch and then merge to the master branch



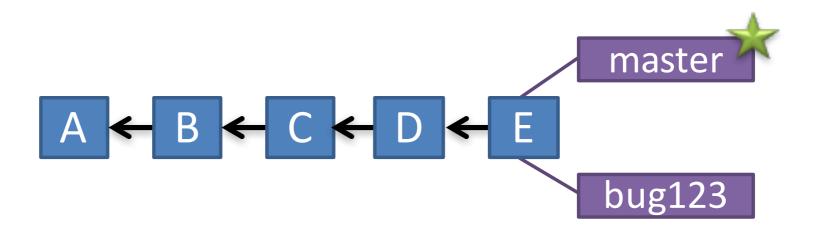
- Branch annotates which commit we are working on
- E.g. we can work on development, create a new branch to handle a bug, write code in the branch and then merge to the master branch



- Branch annotates which commit we are working on
- E.g. we can work on development, create a new branch to handle a bug, write code in the branch and then merge to the master branch



- Branch annotates which commit we are working on
- E.g. we can work on development, create a new branch to handle a bug, write code in the branch and then merge to the master branch





- Use checkout to select a committed version of the project or to branches
 - allows to go back in time, e.g. to see when a bug was introduced
 - we can also just evaluate the difference between current ad older versions of code base

Working with remote

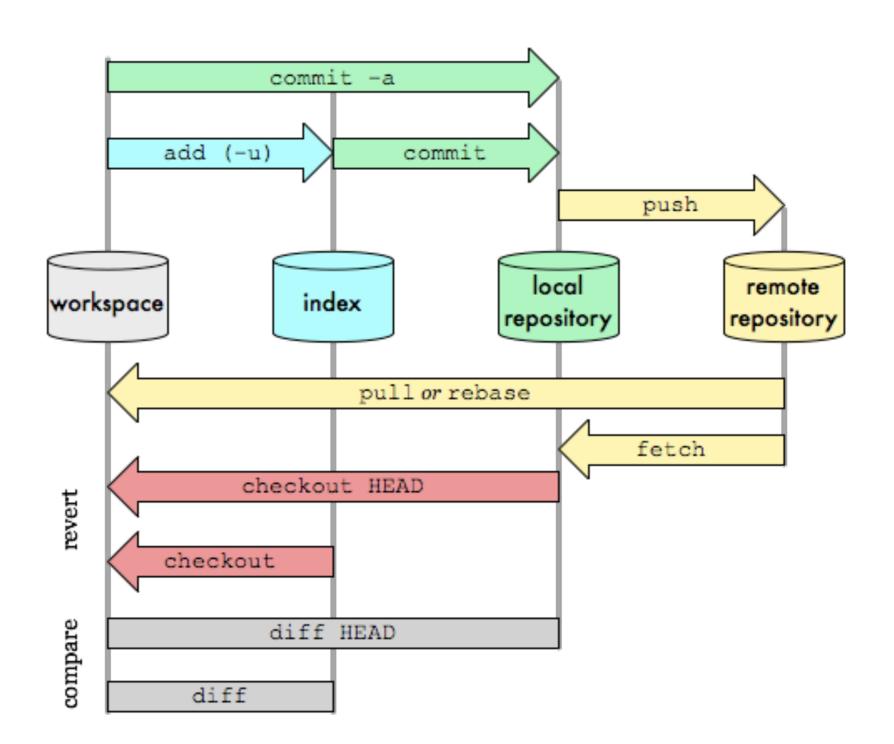
- Add and Commit your changes to your local repo
- **Pull from remote** repo to get most recent changes (fix conflicts if necessary, add and commit them to your local repo)
- **Push** your changes **to** the **remote** repo
- Fetch to retrieve from remote without merging with current code.

Working with remote

- Add and Commit your changes to your local repo
- **Pull from remote** repo to get most recent changes (fix conflicts if necessary, add and commit them to your local repo)
- **Push** your changes **to** the **remote** repo Good practice: Pull then Push

Push will update the remote server. If you are out of date, Git will reject that push.

Git at a glance





Git and CLion

Check git install

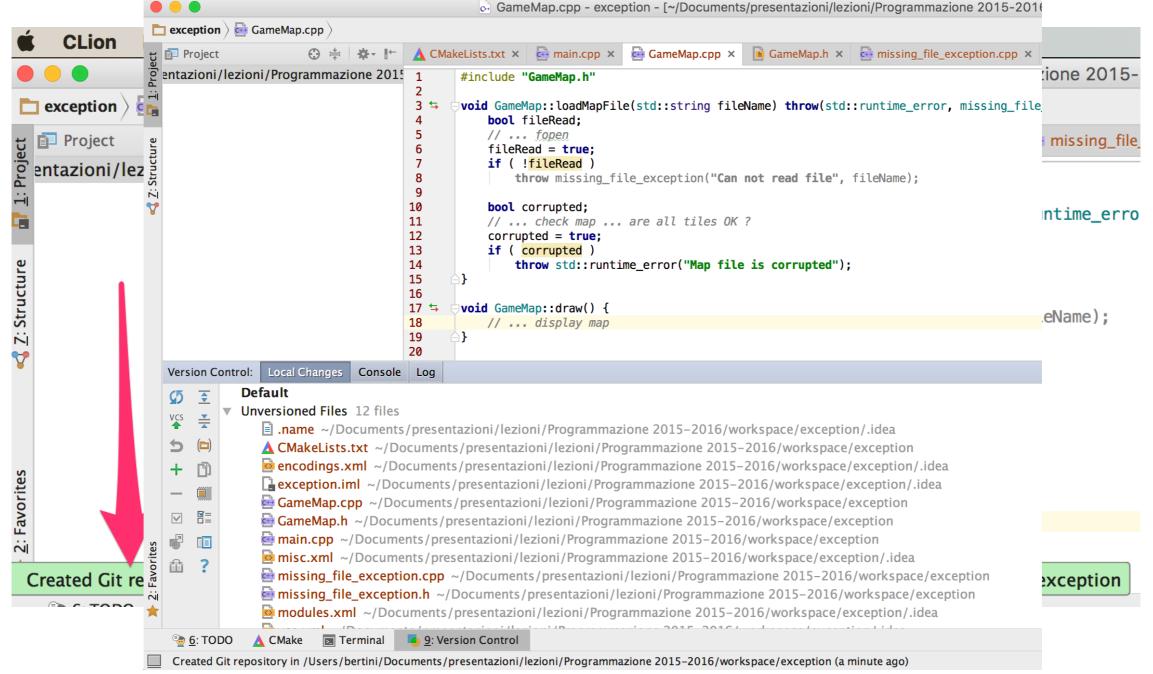
	Preferences
Q Search	Version Control > Git G For current project
 Appearance & Behavior Keymap Editor Plugins Version Control Confirmation Background Ignored Files 	Path to Git executable: /usr/bin/git SSH executable: Built-in Commit automatically on cherry-pick Varn if CRLF line separators are about to be committed Varn when committing in detached HEAD or during rebase Update method: Branch default Auto-update if gush of the current branch was rejected
Issue Navigation Changelist Conflicts GitHub CVS Git Mercurial Perforce Subversion TFS	Allow force push Protected branches: master Test the installed Git

 OSX command line development tools include git. Linux and Windows require to install it.

Ú	CLion	File	Edit	View	Navig	ate Co	ode	Refacto	or Run	Tools	VCS	Window	Help	
						⊶ Game	eMap.c	cpp - exe	ception - [·	~/Docun	n Loca	al History		►
Þ	exception										Ena	ble Versio	n Control Integrati	on
ect	Project				* - I←	🛕 CMa	keLists	.txt ×	🕂 main.cp	p × 🗗	VCS	6 Operation	ns Popup	^ V
<u>1</u> : Project	ntazioni/l	ezioni,	/Progra	mmazio	one 2015	1 2	#inc	lude "G	ameMap.h"	I Contraction of the second se	Арр	ly Patch		
						3 ≒ 4	17		<pre>p::loadMa leRead;</pre>	pFile(s	Che	ckout fron	n Version Control	►
e						5 6		// filoPop	<i>fopen</i> d = true ;				rsion Control	•
🟹 <u>7</u> : Structure						7		if (!f	ileRead)			wse VCS R		•
: Str						8 9		thr	ow missin	g_file_	_excepti	.on("Can n	ot read file", fi	ileName);
						10			<pre>rrupted;</pre>					
						11 12			check map ed = true		re all t	iles OK ?		
						12			rrupted)					
						14					_error("	Map file :	is corrupted");	
						15	≙}			_	_		•	
						16								
						17 与 1 18	F		p::draw()					
						10		//	display m	laμ				
						20	,							

	Enable Version Control Integration	
Select a version control	system to associate with the project root:	
Version control settings	can be configured in 'Settings Version Co	CVS
?	Canc	Git
		Subversion
		TFS
?	Canc	Mercurial Perforce Subversion

Ć	CLion	File	Edit	View	Navigat	e C	ode Re	factor	Run	Tools	VCS	Window	w Help		
					C++	Gam	Мар.срр	- excep	otion - [·	~/Docum	nents/pr	esentazio	ni/lezioni/P	Program	mazione 2015-
C	exception	🛛 🔂 🖓	meMap.c	$\left< qq \right>$											
ect	Project		(⊕ ≑	₩-1-	🔥 СМ	keLists.txt	×	main.cp	p× c+	GameM	ap.cpp ×	📙 GameM	lap.h ×	🖶 missing_file
s 📢 <u>7</u> : Structure 🔐 <u>1</u> : Project		ezioni/	'Progra	mmazio	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	1 2 3 ≒ 5 6 7 8 9 0 1 2 3 4 5	boo // fil if boo // cor	meMap: l file eRead (!fil throw l corr rupted (corr	<pre>:loadMa Read; pen = true; eRead) missin upted; eck map = true upted)</pre>	pFile(s g_file_ ar	except: re all a	ion(" Can tiles OK	not read	file",	<pre>:runtime_erro fileName);</pre>
 <u>2</u>: Favorites 					1 1 1	6 7 ≒ 8 9	void Ga // }		:draw() <mark>splay m</mark>						
(Created Git	reposit	tory in	/Users/	bertini/D	ocum	nts/prese	entazio	ni/lezio	oni/Prog	ramma	zione 201	5-2016/w	orkspac	e/exception



Add files

CLion File Edit	View Navigate Code Refactor	Run Tools VCS Window Help	exception CMakeLists.txt				
	GameMap	.cpp - exception - [~/Documents/presentazioni/lezioni/Pr					
ception 👌 🛕 CMakeLists.tx	rt >		¹ Project ¹ ¹ ¹ ¹ ¹ ¹ ¹ ¹				
Project 😯	🚔 🔅 🖿 🚹 CMakeLists.txt × 📴	main.cpp 🗴 📴 GameMap.cpp 🗴 📑 GameMap.h 🗴 📴 m	Perform and a second secon	2			
CMakeLists.txt	2 3 5 Quoid GameMan	<pre>Map.h" loadMapFile(std::string fileName) throw(std::runt: ead;</pre>	GameMap.cpp GameMap.h GameMap.h GameMap.h GameMap.h GameMap.h GameMap.cpp	<pre>3 \(\mathbf{\Second}\) void GameMap::loadMapF 4 bool fileRead; 5 // fopen 6 fileRead = true; 7 if (!fileRead)</pre>			
GameMap.h main.cpp missing_file_exe	5 Reload CMake Project	en true; Read)	 missing_file_exception.cpp missing_file_exception.h missing_file_exception.h External Libraries 	8 throw missing 9 10 bool corrupted; 11 // check map.			
🝺 missing_file_ex	₭ Cut жX	<pre>missing_file_exception("Can not read file", fileN;</pre>		12 corrupted = true;			
💼 External Libraries	Copy 10 boo #Corr	pted;		13 if (corrupted) 14 throw std::rur			
		ck map are all tiles OK ?		15 _}			
	Conver Diain Toxt	= true; pted)		16 17 ≒ ⊡void GameMap::draw()			
		<pre>std::runtime_error("Map file is corrupted");</pre>		18 // display map			
	් Paste ೫V			19 🔄 } 20			
-	17 🖘 🗇 void GameMap::	draw() {	Version Control: Local Changes Conso	ole Log			
_	Inspect Code	play map					
	Refactor		🚽 💆 🖉 Universioned Files & files				
rsion Control: Local Cha	Clean Python Compiled Files			nts/presentazioni/lezioni/Programn			
5	Add to Favorites		 Image: Second state of the second				
s ₹ Onversioned iname iname iname		rogrammazione 2015-2016/workspace/exception/.ide lezioni/Programmazione 2015-2016/workspace/excep	— 🔳 🔂 modules.xr 🕻 Co	pmmit Changes Province and Province Address Province			
- 🛐 🔂 encodi	Local History	lezioni/Programmazione 2015-2016/workspace/excep	_ຜ 🖥 🔲	ove to Another Changelist			
except	on. Git – (Documenta / presentacio 🕨	cio Check In mazione 2015–2016/workspace/except		ow Diff %D			
🔄 🔤 GameN	5 Synchronize selected files	+ Add \\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\		mp to Source			
📲 🔤 GameN	Poveni in Finder		★ X De	lete			
main.c	Reveal in Finder	Annotate	Sector Ad	ld to VCS			
n ?	Create Gist	Show Current Revision	Schedule selected files to be adde	nore			
📑 missing	g_file_exception.h ~/Documents/pres	Compare with the Same Repository Version Compare with Latest Repository Version Compare with	The ports tree has	eate Patch elve Changes			
	Terminal <u>9</u> : Version Control	Compare with Branch	[Ripristinato: 5 Re	fresh			
			Last login: Thu Ju				

Stage files adding them to git versioning. Use project view or Version Control tab that shows also invisible files like those of the CLion project.

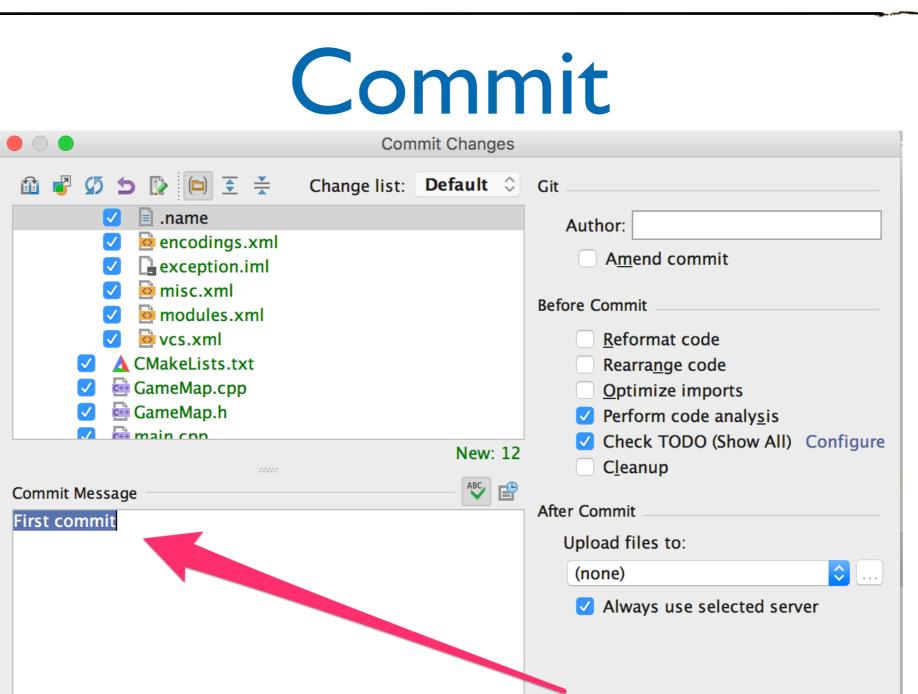
Commit

		⊶ GameMap.cpp - exception - [~/Documents/prese
	exception		,,,
t	Project C) ≑ 🏘 🗜 🔥 CMakeLists.txt × 📴 main.cpp × 📴 Gan	neMap.cpp × 📑 Gai
<u>I</u> : Project	exception ~/Docu	ments/presen 1 #include "ComeMan b"	
1	CMakeLists.txt	New 2	
	🖶 GameMap.cpp 🝺 GameMap.h	PFile(std::	string fileName)
7. Ju ucuire	📴 main.cpp	K Cut	
	🔤 missing_file_ex	Copy #C by file even	ption(" Can not re
б -	imissing_file_exe	Copy Path 企業C	prion (can not re
?	External Libraries	Convas Plain Text	
		Copy Reference Copy	l tiles OK ?
		Paste 13 if (corr #Vd)	
		runtime_erro	or("Map file is co
		Find Usages TF7	
		Find in Path Youd Gamesia 企業Fin() {	
		Replace in Path 企業R ^{nap} Inspect Code	
	Version Control: Local Cha		
	Ø	Refactor	
	▼ T /llsers	Clean Python Compiled Files	ne 2015-2016/wo
	Vcs in posers	Add to Favorites	
		Show Image Thumbnails 企業T	
	+ 🖻 🔤 Gan		
	- 🖬 🔤 Gan		
			Directory
		Synchronize 'exception' + Add	
LILES	💕 📋 🔤 mis	ma_me_exception	

• Commit whole directory or single files

Details

?



Commit

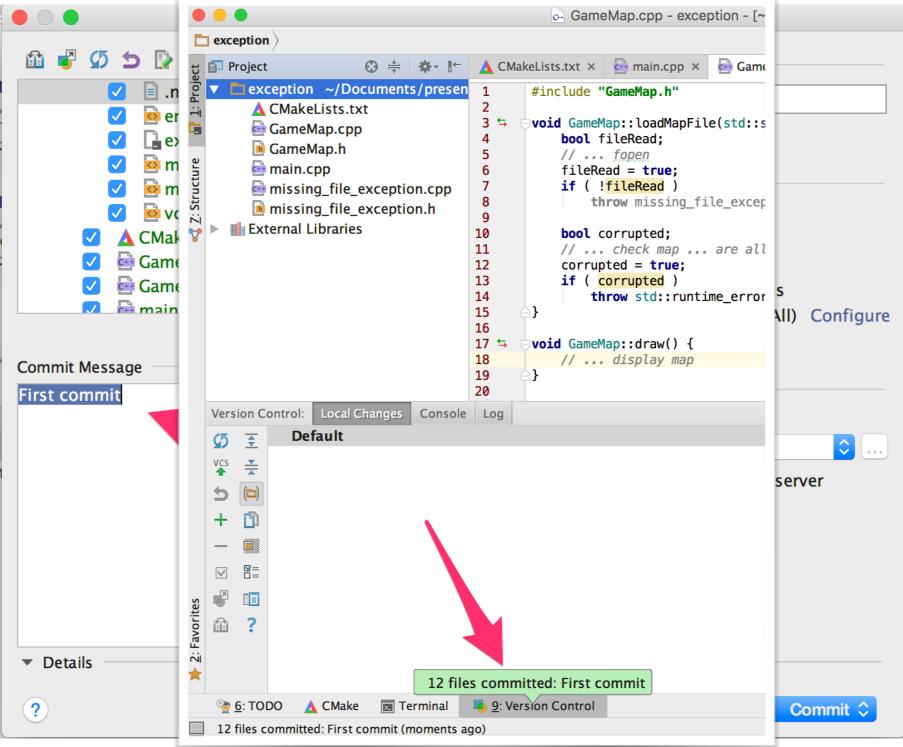
message

Cancel

Commit 🗘

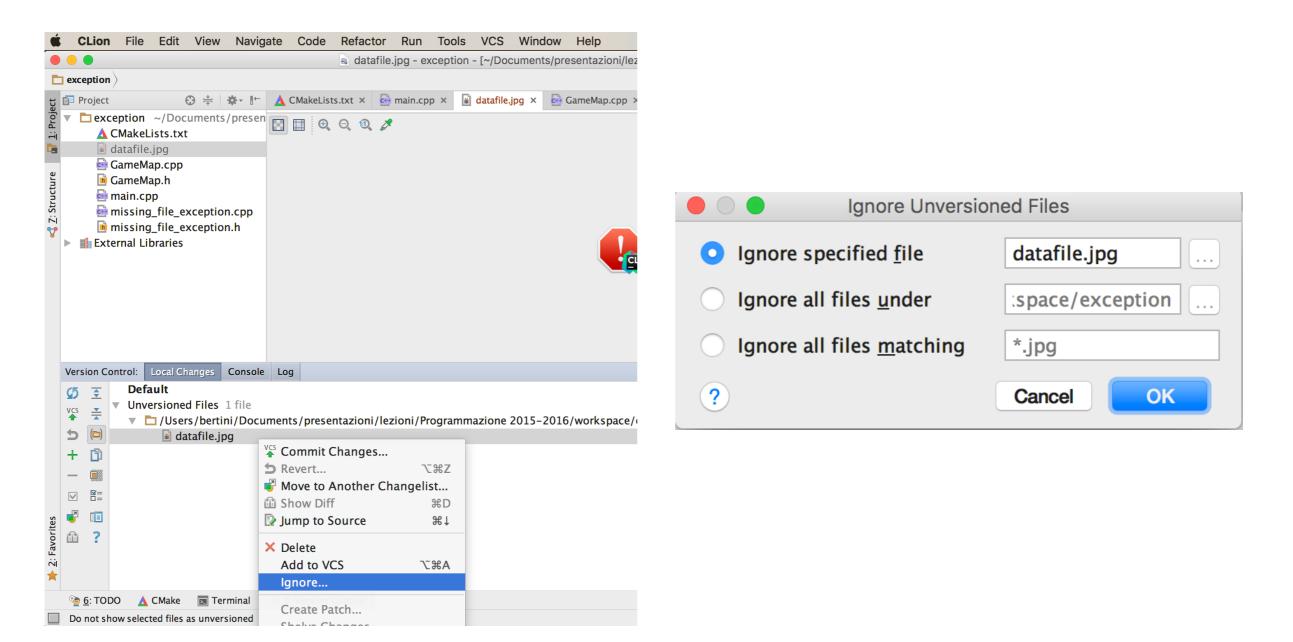
Commit whole directory or single files

Commit



Commit whole directory or single files

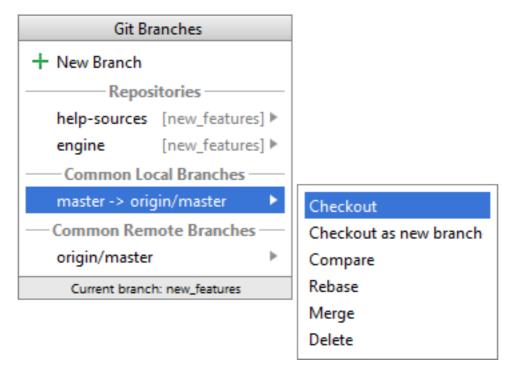




Add to .gitignore with Ignore

Branch

- Use the contextual menu to add new branches, or to checkout them.
- The same menu can be used to merge the current branch with one of the list. The same applies for comparison.



Github

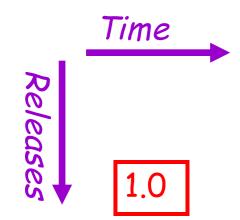
 CLion can use Github as remote server, and allows also to create an account from the options

		Preferences
Q Search	Version Control > GitHub @ For current project Reset	
Appearance & Behavior	Host:	github.com Auth Type: Password
Keymap	Login:	
► Editor	Password:	
Plugins	rassword.	
Version Control		Do not have an account at github.com? Sign up Test
Confirmation @		
Background @	🗹 Clone g	git repositories using ssh
Ignored Files	Connection	timeout: 5.000 🖆 ms
Issue Navigation		
Changelist Conflicts		
GitHub		
CVS @		
Git		
Mercurial @		
Perforce @		add your account or sign up for a new
Subversion @		
TFS		one
Build, Execution, Deployment		
Languages & Frameworks		
► Tools		
cppcheck configuration		



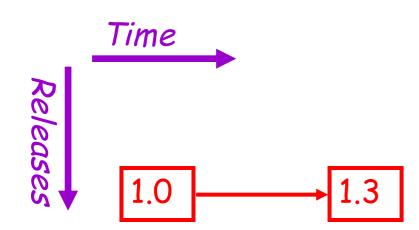
Use scenarios





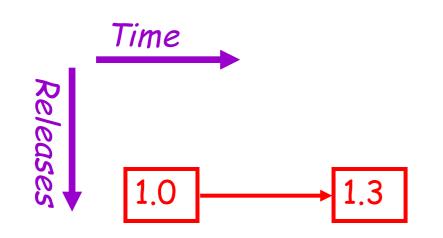
First public release of the hot new product

Scenario I: bug fix



First public release of the hot new product

Scenario I: bug fix



First public release of the hot new product

Internal development continues, progressing to version 1.3

Time

Releases

Scenario I: bug fix

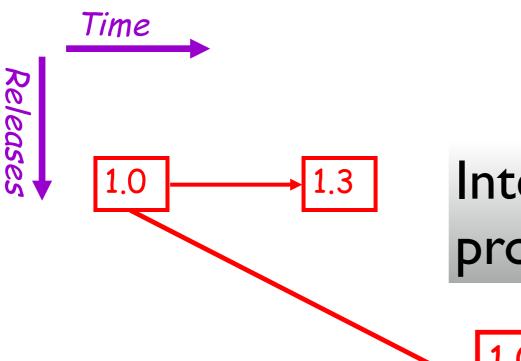
First public release of the hot new product

Internal development continues, progressing to version 1.3



A fatal bug is discovered in the product (1.0), but 1.3 is not stable enough to release. Solution: Create a version based on 1.0 with the bug fix.

Note that there are now two lines of development beginning at 1.0. This is branching.



First public release of the hot new product

Â

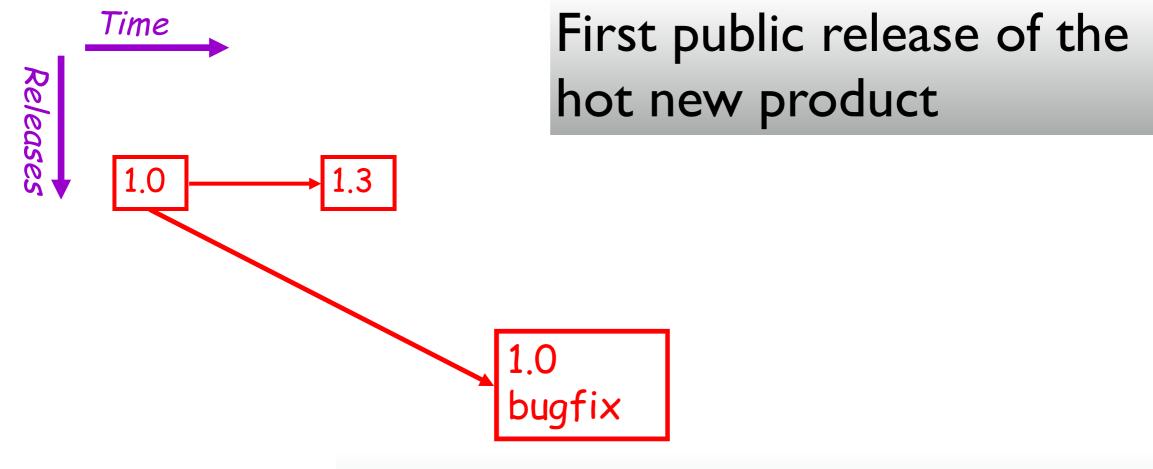
£

Internal development continues, progressing to version 1.3



A fatal bug is discovered in the product (1.0), but 1.3 is not stable enough to release. Solution: Create a version based on 1.0 with the bug fix.

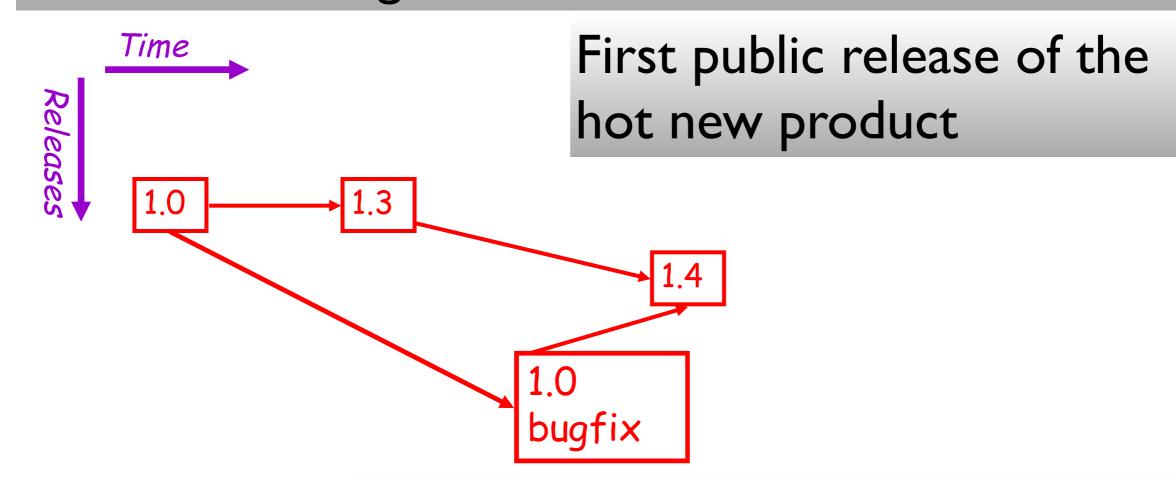
Note that there are now two lines of development beginning at 1.0. This is branching.



A fatal bug is discovered in the product (1.0), but 1.3 is not stable enough to release. Solution: Create a version based on 1.0 with the bug fix.

n 🦨

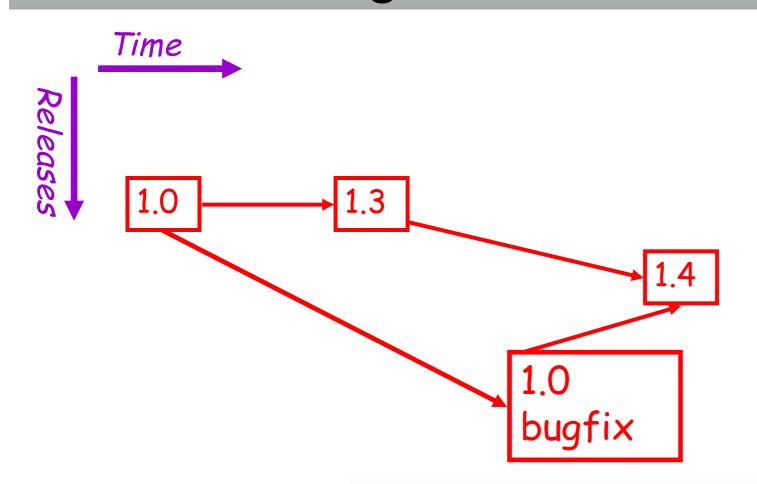
Note that there are now two lines of development beginning at 1.0. This is branching.



A fatal bug is discovered in the product (1.0), but 1.3 is not stable enough to release. Solution: Create a version based on 1.0 with the bug fix.

n 4

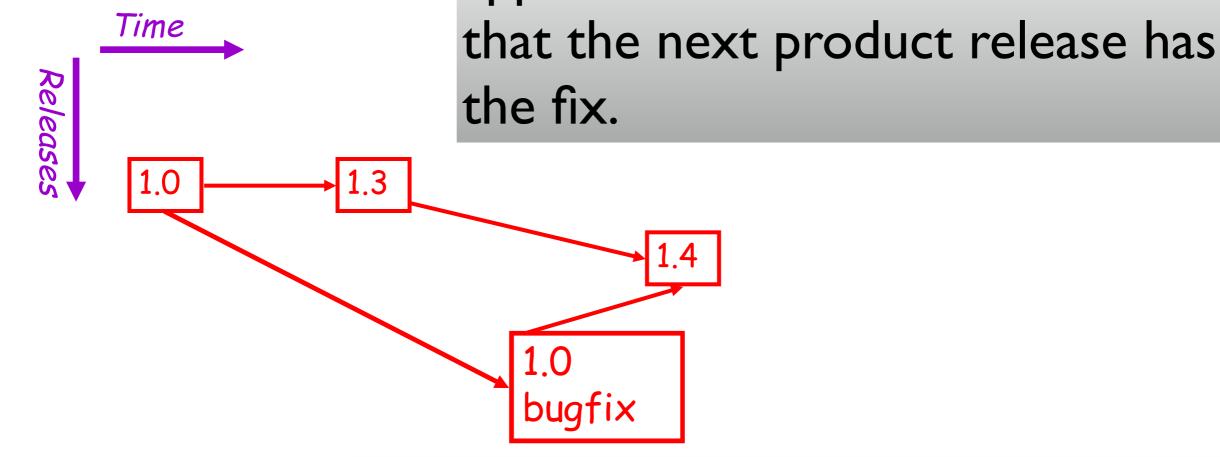
Note that there are now two lines of development beginning at 1.0. This is branching.



A fatal bug is discovered in the product (1.0), but 1.3 is not stable enough to release. Solution: Create a version based on 1.0 with the bug fix.

n 🏠

Note that there are now two lines of development beginning at 1.0. This is branching. The bug fix should also be applied to the main code line so



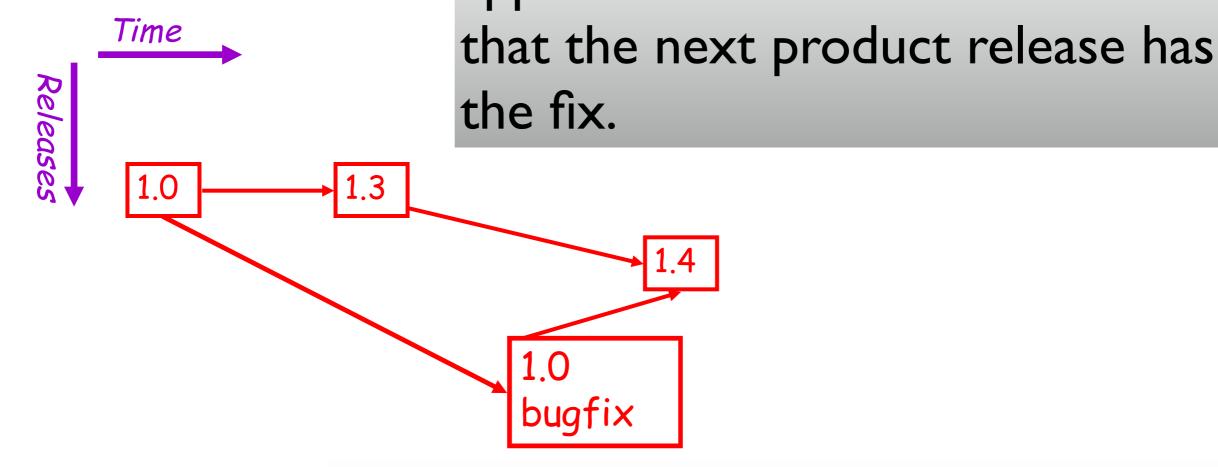
A fatal bug is discovered in the product (1.0), but 1.3 is not stable enough to release. Solution: Create a version based on 1.0 with the bug fix.

n 🗸

Note that there are now two lines of development beginning at 1.0. This is branching. The bug fix should also be applied to the main code line so

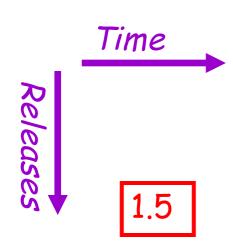
- Car

£

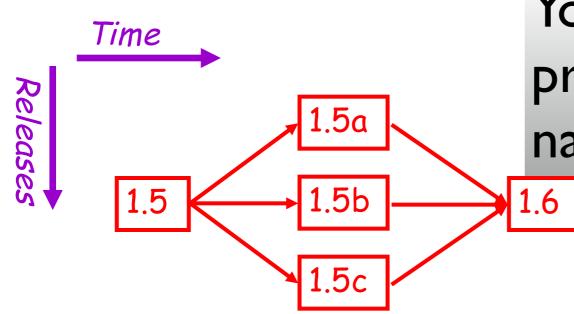


Note that two separate lines of development come back together in 1.4. This is merging.

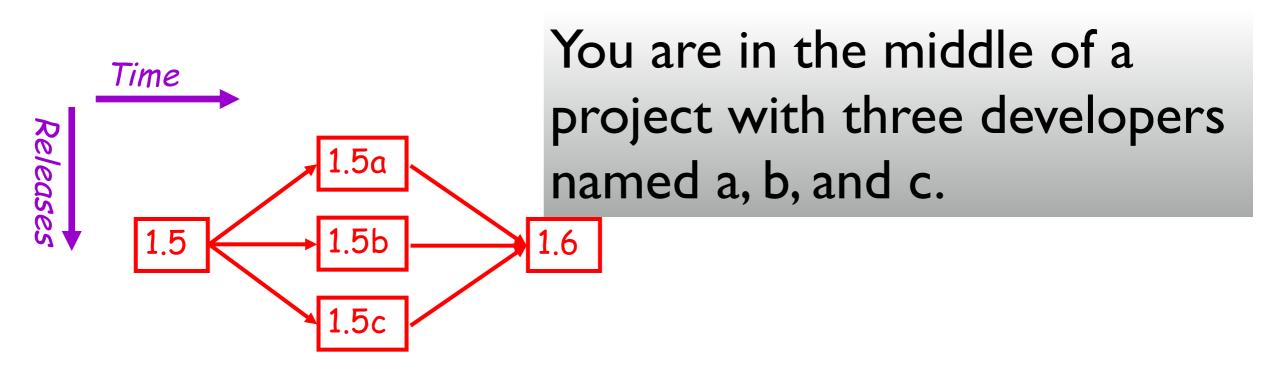
1.6



You are in the middle of a project with three developers named a, b, and c.

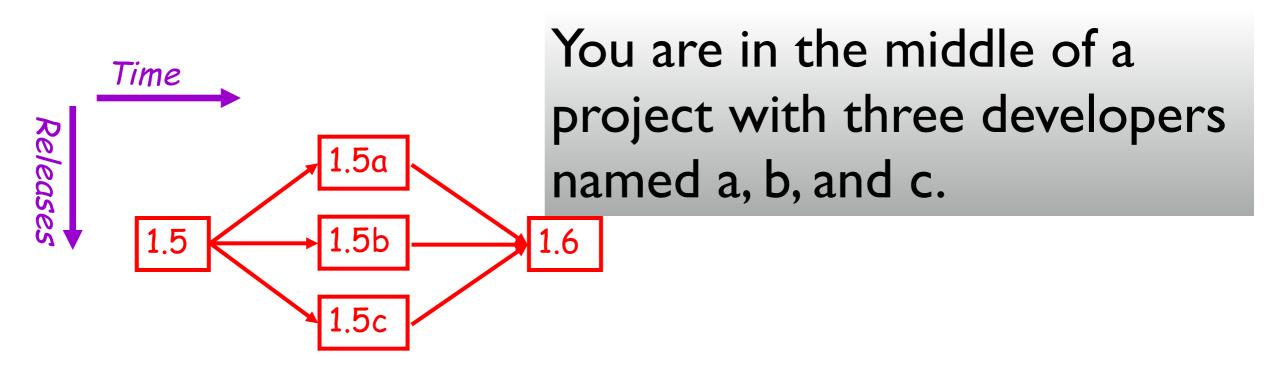


You are in the middle of a project with three developers named a, b, and c.



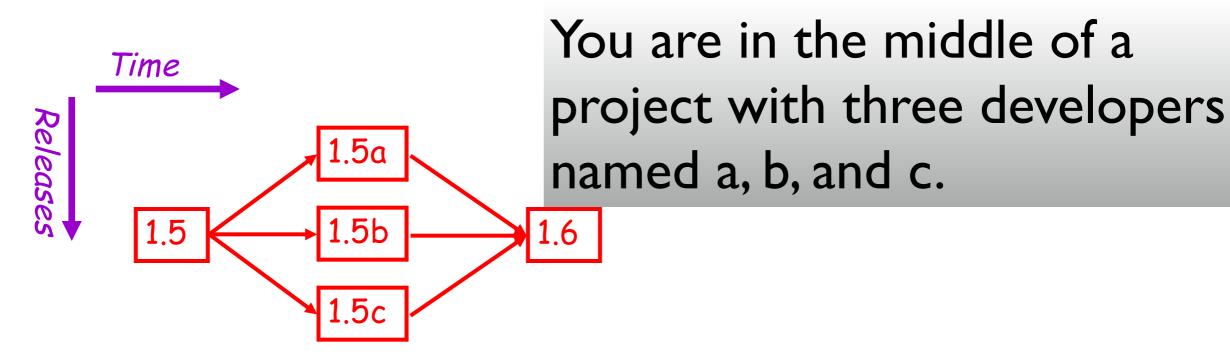
At the beginning of the day everyone checks out a copy of the code.

A check out is a local working copy of a project, outside of the version control system. Logically it is a (special kind of) branch.



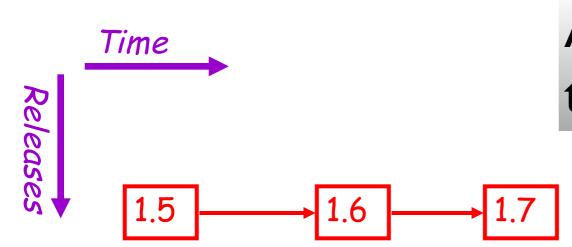
The local versions isolate the developers from each other's possibly unstable changes. Each builds on 1.5, the most recent stable version.

A check out is a local working copy of a project, outside of the version control system. Logically it is a (special kind of) branch.

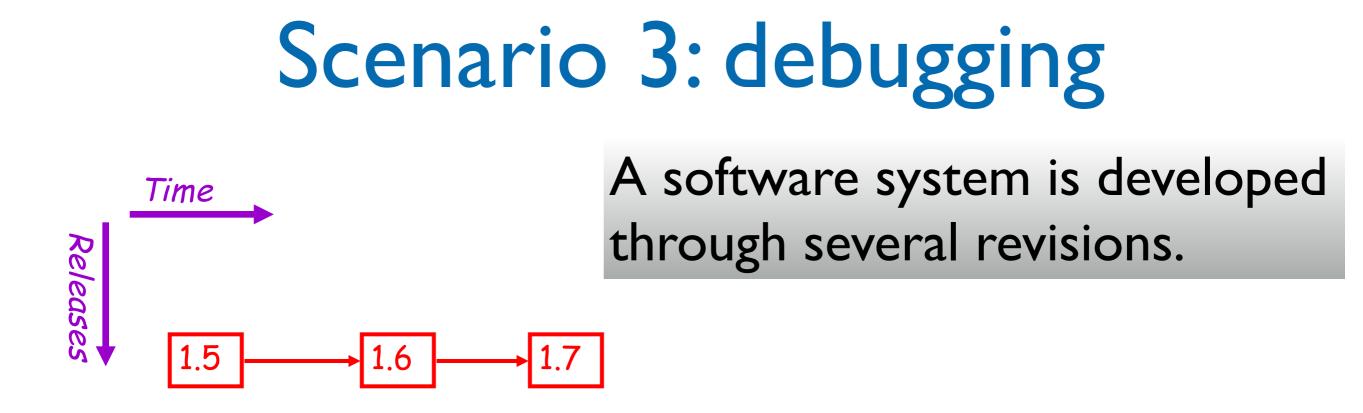


At the end of the day everyone checks in their tested modifications. A check in is a kind of merge where local versions are copied back into the version control system.

Scenario 3: debugging



A software system is developed through several revisions.



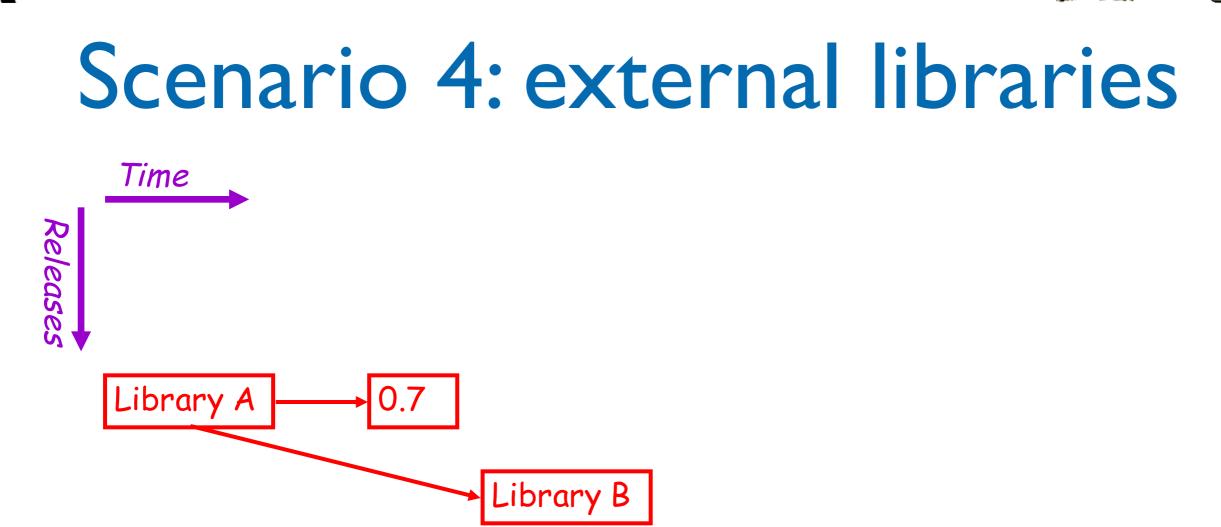
In 1.7 you suddenly discover a bug has crept into the system. When was it introduced?

With version control you can check out old versions of the system and see which revision introduced the bug.



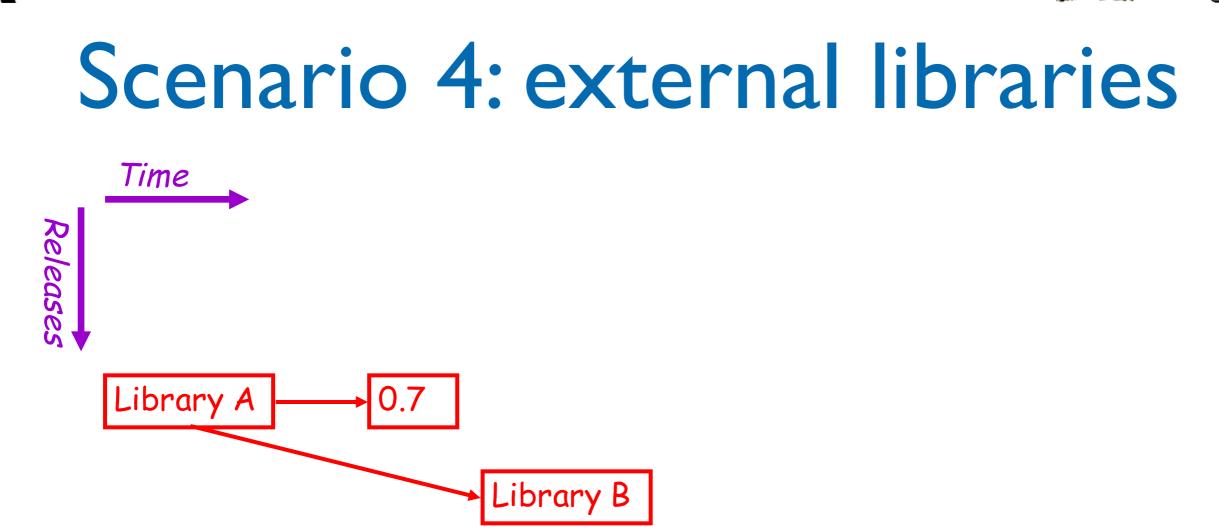
You are building software on top of a thirdparty library, for which you have source.

You begin implementation of your software, including modifications to the library.

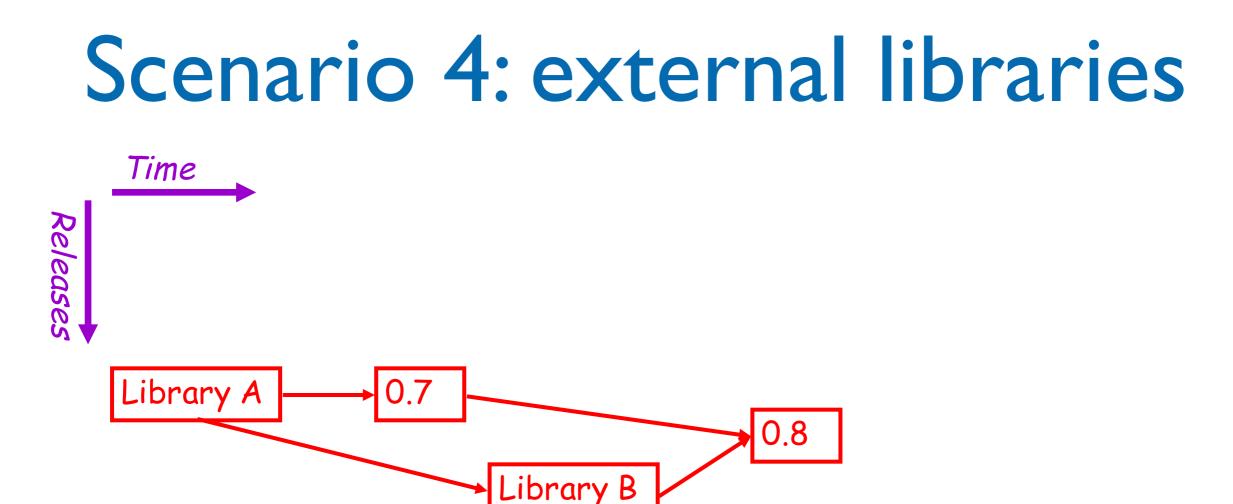


You are building software on top of a thirdparty library, for which you have source.

You begin implementation of your software, including modifications to the library.



A new version of the library is released. Logically this is a branch: library development has proceeded independently of your own development.



You merge the new library into the main code line, thereby applying your modifications to the new library version.



E. Sink, "Version Control by Example" - cap. 2, 4, 8



- These slides are based on the material of:
 - Prof. Aiken,
 - Dr. N. Benatar,
 - Prof. R. Anderson, Univ. Washington
 - P. Chen, Stanford