VCS — Git Support via VC and Magit 🚧

Operation	Keystroke	Function	Note
Emacs Git Support See also: ∑ **VCS-Mercurial** **Magit** • **Cheatsheet** • **Reference Card** • **Magit Homepage** • **Visual walk-through** • **Magical Git Interface**	Emacs supports Git through two different packages: 1. **Emacs VC built-in package**, which has a common UI for all version control system backends. • It works fine with Git and can easily be used for the most common Git commands. Its simple interface is often good enough and simpler to use for simple Git operations like adding/staging files, committing changes, pushing changes to a parent repo. The interface is the same whether you use Git, Mercurial or other VCS supported as VC back-ends. 2. 📦 The **external Magit package**. 🛂 PEL downloads and activates it when the **pel-use-magit** user-option is turned on. • Magit is much more flexible than the VC interface. It's considered one of Emacs killer features. • It helps learn Git features and has extensive support for Git commands.		

Open this PDF file.
See also: ∑ **Help/Info**
	`<f11> v <f1>`
	`<f12> <f1>`
		(**pel-help-pdf** &optional OPEN-WEB-PAGE)	Prompt to open one of the VCS PDF files like the ∑ **VCS-Mercurial** local PDF.
			If the prefix argument (like **`C-u`** or **`M--`**) is used, then it opens the remote GitHub hosted raw PDF instead. If the **pel-flip-help-pdf-arg** user-option is set it's the other way around.

∑ **Customize PEL VCS control**	`<f11> v <f2>`
	`<f12> <f2>`
		(**pel-customize-pel** &optional OTHER-WINDOW)	Customize PEL Version Control System support.
			• If OTHER-WINDOW is non-nil (use **`C-u`**), display in other window.

∑ **Customize Emacs VCS control**	`<f11> v <f3>`	(**pel-customize-library** &optional OTHER-WINDOW)	Customize Emacs Version Control System support: vc, vc-hg, vc-git, magit, monky.
	`<f12> <f3>`		Customize Emacs Version Control System support: vc, vc-hg, vc-git.
Emacs Built-in VCS package: VC	Emacs built-in VCS support is provided by the VC library. • The VC library supports several **VCS tools** identified by the vc-handled-backends, which identifies: **RCS** and **SCCS**, **CVS** and **Subversion**, **Bazaar**, **Git**, **Mercurial** and **Monotone**. • When visiting a directory or a file and using a VC command to manage the repository, VC automatically detects the (D)VCS type for the repository, set the variable *vc-dir-backend* and adjusts its backend accordingly. If the directory has a **.git** directory or is enclosed in a directory tree where the root has a **.git** directory, then VC uses the Git backend and the commands behaves as described in this table.		

• **Forcing selection of a VCS backend for VC commands**
 	
• Select VCS backend with **directory local variable**
  • See also: ∑ **File/Directory Variables**
	The commands in the VC built-in package automatically detect the VCS backend and use it for all their commands.
	• Unfortunately, if a directory tree is managed by more than one VCS tool, the tool selected by the VC commands may not be the one you want. For example if a directory tree is managed both by Git (there is a .git directory in the parent directories) and by Mercurial (and there is a .hg directory in the parent directories) then VC will select Git because its code looks for Git before it looks for Mercurial.
	You can **force the selection** of a specific VCS backend by using one of the following methods:
	1. Force VCS backend for the directory using a directory local variable:
	Write the following in a .dir-locals.el file if the file is empty:
	`((vc-dir-mode . ((vc-dir-backend . Hg))))`
	If the file already contains a list, insert the following entry in the list:
	`(vc-dir-mode . ((vc-dir-backend . Hg)))`
	You might also have to add the safe values of *vc-dir-backend* to the *safe-local-variable*-values, by adding the following statement inside the **custom-set-variables** arguments in your Emacs custom file.:
	`'(safe-local-variable-values`
	`   (quote`
	`    ((vc-dir-backend . Git)`
	`     (vc-dir-backend . Hg))))`
	2. Select a different VCS back-end one file at a time, using the **pel-vcs-switch-backend** command. It searches for the VCS repository directories in the directory parents, then prompt for the one to use and switches to it using the vc-switch-backend command. The vc-switch-backend command can be invoked directory but it does not seem to work when trying to to switch backend in a multi-backend directory environment.
	3. Avoid using VC. Instead use a VCS-specific tool. Like **Magit** for Git and **Monky** for Mercurial.		

Detect available VCS backend and switch to one selected	`<f11> v s`	(**pel-vcs-switch-backend**)	Switch VCS back-end for the current file. • If no VCS back-end or only one VCS back-end is available for the file, the command issues an error, otherwise it prompts for the VCS back-end to use and switch the VCS back-end for this file. The switch is temporary: it is restricted to the current Emacs • session. • Use this command when the file is managed by more than one VCS back-end. • This uses the function 'vc-switch-backend' to perform the switch.
Switch VCS backend	`C-x v b`	(**vc-switch-backend** FILE BACKEND)	Make BACKEND the current version control system for FILE. • FILE must already be registered in BACKEND. The change is not permanent, only for the current session. This function only changes VC's perspective on FILE, it does not register or unregister it. • By default, this command cycles through the registered backends. • To get a prompt, use a prefix argument. ⚠️This command does not seem to be working if a directory is both managed by Mercurial and Git. Use **pel-vcs-switch-backend** instead, it does work.
Control .gitignore	Identify files that must not be managed by Mercurial with the **.gitignore file**.		
Ignore a file (update the .hgignore file)	`C-x v G`	(**vc-ignore** FILE &optional DIRECTORY REMOVE)	Update the depot's **.gitignore** file to ignore a specific file or files. • Ignore FILE under the VCS of DIRECTORY. • Normally, FILE is a wildcard specification that matches the files to be ignored. When REMOVE is non-nil, remove FILE from the list of ignored files. • DIRECTORY defaults to 'default-directory' and is used to determine the responsible VC backend. • When called interactively, prompt for a FILE to ignore, unless a prefix argument is given, in which case prompt for a file FILE to remove from the list of ignored files.
	`G`	(**vc-dir-ignore**)	Ignore the current file; update the .hgignore file.

Operation	Keystroke	Function	Note
VC commands for Git • Git Reference	Basic Git commands are well supported by Emacs VC mode. That works as long as Git is installed and the Emacs process has access to the git commands. • No special setup is required, VC will automatically detect that the file is inside a Git repository as described above. • When editing a file that is inside a Git repository, you can use the following VC commands. • Since the VC command key bindings is the same for all VCS backends, some of the commands do not apply to Git and are not listed below.		
• **git commands:** • git add • git commit	To add a file to a Git repo it must first be staged with the git add command. Then the changes must be committed with the git commit command. • The VC interface uses one command for the two operations. The **vc-register** performs a commit operation if the file is already staged, otherwise it only stages it. • Inside the *vc-dir* buffer this command is mapped to the **v** key. That key will only operate on all marked files as long as the status of each of the marked file is the same.		
Add (stage) a file	`C-x v i`	(**vc-register** &optional VC-FILESET COMMENT)	Register into a version control system: perform a **git add** for the file.
Add/Commit current file	`C-x v v`	(**vc-next-action** VERBOSE)	Executes, for the current file (or all marked files in the *vc-dir* buffer): • **git add** of the files if the files are not part of the repository yet. • **git commit**, of all marked files, otherwise. • Refuses to perform the action when state of *vc-dir* selected items differ.
Add/Commit selected file(s)	`v`		
• **git blame command**	List changes in files, showing the revision id responsible for each line. • This command is useful for discovering when a change was made and by whom.		
Annotate version of each line of file git blame	`C-x v g`	(vc-annotate FILE REV &optional DISPLAY-MODE BUF MOVE-POINT-TO VC-BK)	Annotate the lines of the current file: open a ***Annotate file (rev #)*** buffer that shows the annotation of each line of the current file, each changes has its own background colour. • More commands are available in the *Annotate* buffer.
• **Visit specified revision**	Print the specified files as they were at the given revision.		
Show the content of a specific revision of a file git cat-file	`C-x v ~`	(**vc-revision-other-window** REV)	Visit revision REV of the current file in another window. • If the current file is named ‘F’, the revision is named ‘F.~REV~’. • If ‘F.~REV~’ already exists, use it instead of checking it out again.
• **git diff command**	Show differences between revisions for the specified files.		
Diff versions of all files in directory: git diff	`C-x v D`	(**vc-root-diff** HISTORIC &optional NOT-URGENT)	Executes: **git diff** command to list differences between all files in the local directory and the repository. • Display diffs between VC-controlled whole tree revisions. • Normally, this compares the tree corresponding to the current fileset with the working revision. • With a prefix argument HISTORIC, prompt for two revision designators specifying which revisions to compare. • The optional argument NOT-URGENT non-nil means it is ok to say no to saving the buffer.

Diff versions of current file:
git diff
	`C-x v =`
	`=`
		(**vc-diff** &optional HISTORIC NOT-URGENT)	Executes: **git diff** command to list differences between current file version in repo and local file’s content.
			• Display diffs between file revisions.
			• Normally this compares the currently selected fileset with their working revisions. With a prefix argument HISTORIC, it reads two revision designators specifying which revisions to compare.
			• The optional argument NOT-URGENT non-nil means it is ok to say no to saving the buffer.

Diff currently selected file(s) git diff	`d` `D`	(**log-view-diff** BEG END) (**log-view-diff-changeset** BEG END)	Show the file(s) difference(s) for the log entry inside a *vc-diff* buffer. • Get the diff between two revisions. • If the region is inactive or the mark is on the revision at point, get the diff between the revision at point and its previous revision. Otherwise, get the diff between the revisions where the region starts and ends. • Unlike ‘log-view-diff-changeset’, this function only shows the part of the changeset which affected the currently considered file(s).
• **Incoming changes**	Show new changesets found in the specified path/URL or the default pull location. • These are the changesets that would have been pulled by hg pull at the time you issued this command.		

List files incoming from parent (or specified) repo	`C-x v I`
	`I`
		(**vc-log-incoming** &optional REMOTE-LOCATION)	List files incoming from parent (or specified) repo in the *vc-incoming* buffer.
			• Show a log of changes that will be received with a pull operation from REMOTE-LOCATION.
			• When called interactively with a prefix argument, prompt for REMOTE-LOCATION.

• **git log command**	Print the revision history of the specified files or the entire project. • Open a ***vc-change-log*** window buffer, using the vc-hg-log-view-mode major mode which provides other commands. See section below.		
Print repo log git log	`C-x v L`	(**vc-print-root-log** &optional LIMIT)	Executes: **git log** to list the repo log in a ***vc-change-log*** buffer. • List the change log for the current VC controlled tree in a window. • If LIMIT is non-nil, it should be a number specifying the maximum number of revisions to show; the default is ‘vc-log-show-limit’. • When called interactively with a prefix argument, prompt for LIMIT.

Open the change log for the file
git log
	`C-x v l`
	`l`
		(**vc-print-log** &optional WORKING-REVISION LIMIT)	Executes: **git log.** Open the change log for the file in the ***vc-change-log*** buffer.
			• List the change log of the current fileset in a window.

• **git merge command** • git mergetool	Merge another revision into working directory.		
Merge git merge	`C-x v m`	(**vc-merge**)	Executes: **git merge**. Perform a version control merge operation. • You must be visiting a version controlled file, or in a ‘vc-dir’ buffer. • This runs a “merge" operation to incorporate changes from another branch onto the current branch, prompting for an argument list.
• **Show outgoing**	Show changesets not found in the destination.		

Show change sets not found in the destination	`C-x v O`
	`O`
		(**vc-log-outgoing** &optional REMOTE-LOCATION)	List deltas to push to parent repo.
			• Show a log of changes that will be sent with a push operation to REMOTE-LOCATION.
			• When called interactively with a prefix argument, prompt for REMOTE-LOCATION.

• **git pull command**	Pull changes from a remote repository to a local one.		
Pull files from parent repo: git pull	`C-x v +`	(**vc-update** &optional ARG)	Executes: **git pull** command to pull files from parent repository. • Update the current fileset or branch. • You must be visiting a version controlled file, or in a ‘vc-dir’ buffer. • On a distributed version control system, this runs a “pull" operation to update the current branch, prompting for an argument list if required. • Optional prefix ARG forces a prompt for the VCS command to run, allowing the addition of command line options.

Operation	Keystroke	Function	Note
• **git push command**	Push changesets from the local repository to the specified destination.		

Push changes to the specified destination
 	
git push
	`C-x v P`
	`P`
		(**vc-push** &optional ARG)	Executes: **git push** to push committed change sets to the parent repo.
			• Push the current branch.
			• You must be visiting a version controlled file, or in a 'vc-dir' buffer.
			• On a distributed version control system, this runs a "push" operation on the current branch, prompting for the precise command if required. Optional prefix ARG non-nil forces a prompt for the VCS command to run.

• **git rm command**	Schedule the indicated files for removal from the current branch.		
Delete a file git rm	`C-x v x`	(**vc-delete-file** FILE)	Executes: **git rm** Delete file and mark it as such in the version control system. Prompts for the file name, using the directory of the file visited in current buffer.
• **git mv command**	Move or rename a file, a directory, or a symlink		
Rename a file hg rename	`M-x vc-rename-file`	(**vc-rename-file** OLD NEW)	Rename file OLD to NEW in both work area and repository. • If called interactively, read OLD and NEW, defaulting OLD to the current buffer's file name if it's under version control.
• **Restore file**	Restore files to their checkout state (the last committed version by default).		
Revert file(s) hg revert	`C-x v u`	(**vc-revert**)	Revert working copies of the selected fileset to their repository contents. • This asks for confirmation if the buffer contents are not identical to the working revision (except for keyword expansion).
• **git tag command**	Name a particular revision.		

Create a tag
 	
git tag
	`C-x v s`
	`B c`
		(**vc-create-tag** DIR NAME BRANCHP)	Executes: **git tag**. Descending recursively from DIR, make a tag called NAME.
			• For each registered file, the working revision becomes part of the named configuration. If the prefix argument BRANCHP is given, the tag is made as a new branch and the files are checked out in that new branch.

Retrieve a tagged version	`C-x v r`
	`B s`
		(**vc-retrieve-tag** DIR NAME)	For each file in or below DIR, retrieve their tagged version NAME.
			• NAME can name a branch, in which case this command will switch to the named branch in the directory DIR.
			• Interactively, prompt for DIR only for VCS that works at file level; otherwise use the repository root of the current buffer.
			• If NAME is empty, it refers to the latest revisions of the current branch.
			• If locking is used for the files in DIR, then there must not be any locked files at or below DIR (but if NAME is empty, locked files are allowed and simply skipped).
			• Tab completion on tag is available at the prompt.
			⚠️ Tags show in tab completion do not include the tags created with git tag. Instead they only show the changeset short ID number. However, it accepts names of tags previously created with git tag.
			• You can list the tags with the command **git tag** on the command line.

Show the change log	`B l`	(**vc-print-branch-log** BRANCH)	Show the change log for BRANCH in a *vc-change-log* window. • Prompt for the branch name. Tab completion shows the list of changeset short ID numbers. It does not show them but you can also use the names of tags previously created with git tag. • You can list the tags with the command **git tag** on the command line.
***vc-dir* buffer**	The **vc-dir** command opens a *vc-dir* buffer to show the status of files unregistered, modified and not yet committed. The buffer supports a set of commands, mostly single character commands to quickly perform operations on the repository. • Some are shown in the mode-specific key bindings show above. • Some commands are not tied to git commands. They are shown below.		
Show help for the mode	• `?` • `h`	(**describe-mode** &optional BUFFER)	Opens the *Help* buffer with information about the mode, listing the various key bindings and commands available
Refresh buffer	`g`	(**revert-buffer** &optional IGNORE-AUTO NOCONFIRM PRESERVE-MODES)	Restore buffer to the default list view.
Mark file	`m`	(**vc-dir-mark**)	Mark the current file or all files in the region. • If the region is active, mark all the files in the region. Otherwise mark the file on the current line and move to the next line. • Operations from the buffer apply to all marked files, like adding or committing files, or other operations. 💡 This is really useful to commit several files in the same changeset.
Unmark file	`u`	(**vc-dir-unmark**)	Unmark the current file or all files in the region. • If the region is active, unmark all the files in the region. Otherwise unmark the file on the current line and move to the next line.
Unmark all files	• `U` • `M-<DEL>`	(**vc-dir-unmark-all-files** ARG)	Unmark all files with the same state as the current one. • With a prefix argument unmark all files. • If the current entry is a directory, unmark all the child files. • The commands operate on files that are on the same state. • This command is intended to make it easy to deselect all files that share the same state. ☝ The cursor must be located on the line of a marked or updated item.
Pull from parent repo git pull	`+`	(**vc-update** &optional ARG)	Execute: **git pull** Update the current fileset or branch. • Optional prefix ARG forces a prompt for the Mercurial command to run.
Hide items	`x`	(**vc-dir-hide-up-to-date** &optional STATE)	Hide items that are in STATE from display. • Hitting **x** alone hides both 'up-to-date' and 'ignored' items. • To hide a specific file status, move point to a file with that status and then hit `C-u x` while on that line. • To view the files again, hit **g**. ⚠️ Unfortunately the current implementation never shows the up-to-date and ignored items again at least with the implementation as of early 2020. There is a bug on Emacs for this and it was closed without action in October 2019. The reason was they did not see a good reason for fixing it… Well.. being able to open the file from the vc-dir buffer would be one reason…
Close the window	`q`	(**quit-window** &optional KILL WINDOW)	Quit WINDOW and bury its buffer.

Operation	Keystroke	Function	Note
Commands available in the *vc-change-log* buffer	The `C-x v L` and `C-x v l` commands list the repo change log into the ***vc-change-log*** buffer. The following commands are available in the buffer. ⚠️Note that some other commands that are available (like **e** to modify the change comment) do not work for the Mercurial back-end and navigation commands. Use the **describe-mode** command to get more information.		
Show help for the mode	• `?` • `h`	(**describe-mode** &optional BUFFER)	Opens the *Help* buffer with information about the mode, listing the various key bindings and commands available
Toggle log line entry view	`RET`	(**log-view-toggle-entry-display**)	Toggle the view of the log line between a single line to a multi-line with more details.
Show diff for log entry	`=`	(**vc-diff** &optional HISTORIC NOT-URGENT)	
Visit the content of the revision at point.	`f`	(**log-view-find-revision** POS)	Visit the version at POS. If called interactively, visit the version at point.
Mark log entry for future diff operation.	`m`	(**log-view-toggle-mark-entry**)	Mark the current log entry line to use as the version to participate in a diff operation. • Toggle the marked state for the log entry at point. • Individual log entries can be marked and unmarked. The marked entries are denoted by changing their background color. • 'log-view-get-marked' returns the list of tags for the marked log entries.
Close the window	`q`	(**quit-window** &optional KILL WINDOW)	Quit WINDOW and bury its buffer.
***Annotate* buffers** **git blame**	The C-x v g command opens an ***annotate*** buffer where the file revision annotation is shown. The following single key commands are available in this buffer. Use the **describe-mode** command to get more information.		
Show help for the mode	• `?` • `h`	(**describe-mode** &optional BUFFER)	Opens the *Help* buffer with information about the mode, listing the various key bindings and commands available
Show the diff of the revision of the current annotated line	• `d` • `=`	(**vc-annotate-show-diff-revision-at-line**)	Visit the diff of the revision at line from its previous revision. • Open the changes diff in the ***vc-diff*** buffer.
Show the change set corresponding to the annotated line.	`D`	(**vc-annotate-show-changeset-diff-revision-at-line**)	Visit the diff of the revision at line from its previous revision for all files in the changeset. • Open the changes diff in the ***vc-diff*** buffer.
Open the file for the revision that corresponds to the revision of the current annotated line	`f`	(**vc-annotate-find-revision-at-line**)	Visit the file at the revision identified in the current line. • The file at specific revision is opened in a buffer that has the same file name with a suffix that identifies the revision number.
Open the log of the revision of the current annotated line	`l`	(**vc-annotate-show-log-revision-at-line**)	Visit the log of the revision at line; only show the log entry for the revision. • If a *vc-change-log* buffer exists and already shows a log for the file in question, search for the log entry required and move point.
Update the annotated view to the revision as it was at the version of the current annotated line	`j`	(**vc-annotate-revision-at-line**)	Visit the annotation of the revision identified in the current line.
Update the annotated view to the next revision	`n`	(**vc-annotate-next-revision** PREFIX)	Visit the annotation of the revision after this one. • With a numeric prefix argument, annotate the revision that many revisions after.
Update the annotated view to the previous revision	`p`	(**vc-annotate-prev-revision** PREFIX)	Visit the annotation of the revision previous to this one. • With a numeric prefix argument, annotate the revision that many revisions previous.
Update the annotated view to the current working version	`w`	(**vc-annotate-working-revision**)	Visit the annotation of the working revision of this file.
Close the window	`q`	(**quit-window** &optional KILL WINDOW)	Quit WINDOW and bury its buffer.
Using Magit **Magit** • **Cheatsheet** • **Reference Card** • **Magit Homepage** • **Visual walk-through** • **Magical Git Interface**	📦 Requires the **external Magit package**. ☑ PEL downloads and activates it when the **pel-use-magit** user-option is turned on. • Magit is much more flexible than the VC interface. It's considered one of Emacs killer features. • It helps learn Git features and has extensive support for Git commands. 🚧 Magit documentation is extensive and excellent. For now this is only showing the PEL specific key used to start magit.		
Start Magit, show Git status of current repository.	`<f11> v g s`	(**magit-status** &optional DIRECTORY CACHE)	Show the status of the current Git repository in a buffer. • If the current directory isn't located within a Git repository, then prompt for an existing repository or an arbitrary directory, depending on option 'magit-repository-directories', and show the status of the selected repository instead. • If that option specifies any existing repositories, then offer those for completion and show the status buffer for the selected one. • Otherwise read an arbitrary directory using regular file-name completion. If the selected directory is the top-level of an existing working tree, then show the status buffer for that. • Otherwise offer to initialize the selected directory as a new repository. After creating the repository show its status buffer. • These fallback behaviors can also be forced using one or more prefix arguments: • With two prefix arguments (or more precisely a numeric prefix value of 16 or greater) read an arbitrary directory and act on it as described above. The same could be accomplished using the command 'magit-init'. • With a single prefix argument read an existing repository, or if none can be found based on 'magit-repository-directories', then fall back to the same behavior as with two prefix arguments.

VCS - Git — Reference

Topic/URL	Comment
The Pro Git Book	Version 2 of the Pro Git Book, with a link to the table of contents. An essential reference to learn Git.
Using Git with Github	A set of guides for using Git and Github. The link on next row is part of this guide.
Getting started with Git and GitHub	Read this page to learn how to identify yourself and setting credentials for Git using Github so that you won't have to enter your user name and password on every interaction with Github. • On macOS, the Git credentials for Github can be stored in the OSX Keychain . • To store Github credential in the keychain just clone a Github repo the "git clone" with the https URL: macOS will prompt for accessing the Keychain: you must then enter your system password.
Magit	Magit Manual