Sage Reference Manual: Sage's Development Scripts

Release 6.3

The Sage Development Team

August 11, 2014

CONTENTS

CHAPTER

ONE

SAGEDEV

This module provides SageDev, the central object of the developer scripts for sage.

AUTHORS:

- David Roe, Frej Drejhammar, Julian Rueth, Martin Raum, Nicolas M. Thiery, R. Andrew Ohana, Robert Bradshaw, Timo Kluck: initial version

class sage.dev.sagedev.**SageDev**(*config=None*, *UI=None*, *trac=None*, *git=None*)
Bases: sage.dev.patch.MercurialPatchMixin

The developer interface for sage.

This class facilitates access to git and trac.

INPUT:

- config – a Config or None (default: None), the configuration of this object; the defaults uses the configuration stored in DOT_SAGE/devrc.
- UI – a UserInterface or None (default: None), the default creates a cmd_line_interface.CmdLineInterface from config['UI'].
- trac – a trac_interface.TracInterface or None (default: None), the default creates a trac_interface.TracInterface from config['trac'].
- git – a git_interface.GitInterface or None (default: None), the default creates a git_interface.GitInterface from config['git'].

EXAMPLES:

```
sage: dev._sagedev
SageDev()
```

abandon(*ticket_or_branch*, *helpful=True*)
Abandon a ticket or branch.

INPUT:

- ticket_or_branch – an integer or string identifying a ticket or the name of a local branch, remove the branch ticket_or_branch or the branch for the ticket ticket_or_branch. Also removes the users remote tracking branch.
- helpful – boolean (default: True). Whether to print informational messages to guide new users.

See Also:

- prune_tickets() – abandon tickets that have been closed.
- tickets() – list local non-abandoned tickets.

TESTS:

Create a single user for doctesting:

```
sage: from sage.dev.test.sagedev import single_user_setup
sage: dev, config, UI, server = single_user_setup()
```

Create a ticket branch and abandon it:

```
sage: UI.append("Summary: summary\ndescription")
sage: dev.create_ticket()
Created ticket #1 at https://trac.sagemath.org/1.

#  (use "sage --dev checkout --ticket=1" to create a new local branch)
1
sage: dev.checkout(ticket=1)
On ticket #1 with associated local branch "ticket/1".

#  Use "sage --dev merge" to include another ticket/branch.
#  Use "sage --dev commit" to save changes into a new commit.
sage: UI.append("y")
sage: dev.push()
The branch "u/doctest/ticket/1" does not exist on the remote server.
Create new remote branch? [Yes/no] y
sage: dev.abandon(1)
Cannot delete "ticket/1": is the current branch.

#  (use "sage --dev vanilla" to switch to the master branch)
sage: dev.vanilla()
sage: dev.abandon(1)
Moved your branch "ticket/1" to "trash/ticket/1".

#  Use "sage --dev checkout --ticket=1 --base=master" to restart working on #1 with a clean
```

Start to work on a new branch for this ticket:

```
sage: from sage.dev.sagedev import MASTER_BRANCH
sage: UI.append("y")
sage: dev.checkout(ticket=1, base=MASTER_BRANCH)
About to create a new branch for #1 based on "master". However, the trac ticket
for #1 already refers to the branch "u/doctest/ticket/1". The new branch will
not contain any work that has already been done on "u/doctest/ticket/1".
Create fresh branch? [yes/No] y
On ticket #1 with associated local branch "ticket/1".

#  Use "sage --dev merge" to include another ticket/branch.
#  Use "sage --dev commit" to save changes into a new commit.
```

Verify that trac ticket #16249 has been resolved:

```
sage: dev.abandon()
Traceback (most recent call last):
...
TypeError: abandon() takes at least 2 arguments (1 given)
```

browse_ticket(*ticket=None*)

Start a webbrowser at the ticket page on trac.

INPUT:

- `ticket` – an integer or string identifying a ticket or `None` (default: `None`), the number of the ticket

to edit. If None, browse the _current_ticket().

See Also:

edit_ticket(), comment(), sage.dev.trac_interface.TracInterface.show_ticket(), sage.dev.trac_interface.TracInterface.show_comments()

EXAMPLES:

```
sage: dev.browse_ticket(10000) # not tested
```

checkout (*ticket=None*, *branch=None*, *base=''*)

Checkout another branch.

If ticket is specified, and branch is an existing local branch, then ticket will be associated to it, and branch will be checked out into the working directory. Otherwise, if there is no local branch for ticket, the branch specified on trac will be pulled to branch unless base is set to something other than the empty string ''. If the trac ticket does not specify a branch yet or if base is not the empty string, then a new one will be created from base (per default, the master branch).

If ticket is not specified, then checkout the local branch branch into the working directory.

INPUT:

- ticket – a string or an integer identifying a ticket or None (default: None)
- branch – a string, the name of a local branch; if ticket is specified, then this defaults to ticket/ticket.
- base – a string or None, a branch on which to base a new branch if one is going to be created (default: the empty string '' to create the new branch from the master branch), or a ticket; if base is set to None, then the current ticket is used. If base is a ticket, then the corresponding dependency will be added. Must be '' if ticket is not specified.

See Also:

pull(), create_ticket(), vanilla()

TESTS:

Set up a single user for doctesting:

```
sage: from sage.dev.test.sagedev import single_user_setup
sage: dev, config, UI, server = single_user_setup()
```

Create a few branches:

```
sage: dev.git.silent.branch("branch1")
sage: dev.git.silent.branch("branch2")
```

Checking out a branch:

```
sage: dev.checkout(branch="branch1")
On local branch "branch1" without associated ticket.

#  Use "sage --dev merge" to include another ticket/branch.
#  Use "sage --dev commit" to save changes into a new commit.
sage: dev.git.current_branch()
'branch1'
```

Create a ticket and checkout a branch for it:

```
sage: UI.append("Summary: summary\ndescription")
sage: dev.create_ticket()
```

```
Created ticket #1 at https://trac.sagemath.org/1.

#  (use "sage --dev checkout --ticket=1" to create a new local branch)
1
sage: dev.checkout(ticket=1)
On ticket #1 with associated local branch "ticket/1".

#  Use "sage --dev merge" to include another ticket/branch.
#  Use "sage --dev commit" to save changes into a new commit.
sage: dev.git.current_branch()
'ticket/1'
```

checkout_branch(*branch*, *helpful=True*)

Checkout to the local branch `branch`.

INPUT:

- `branch` – a string, the name of a local branch

TESTS:

Set up a single user for doctesting:

```
sage: from sage.dev.test.sagedev import single_user_setup
sage: dev, config, UI, server = single_user_setup()
```

Create a few branches:

```
sage: dev.git.silent.branch("branch1")
sage: dev.git.silent.branch("branch2")
```

Checking out a branch:

```
sage: dev.checkout(branch="branch1")
On local branch "branch1" without associated ticket.

#  Use "sage --dev merge" to include another ticket/branch.
#  Use "sage --dev commit" to save changes into a new commit.
sage: dev.git.current_branch()
'branch1'
```

The branch must exist:

```
sage: dev.checkout(branch="branch3")
Branch "branch3" does not exist locally.

#  (use "sage --dev tickets" to list local branches)
```

Checking out branches with untracked files:

```
sage: open("untracked", "w").close()
sage: dev.checkout(branch="branch2")
On local branch "branch2" without associated ticket.

#  Use "sage --dev merge" to include another ticket/branch.
#  Use "sage --dev commit" to save changes into a new commit.
```

Checking out a branch with uncommitted changes:

```
sage: open("tracked", "w").close()
sage: dev.git.silent.add("tracked")
sage: dev.git.silent.commit(message="added tracked")
```

```
sage: with open("tracked", "w") as f: f.write("foo")
sage: UI.append("cancel")
sage: dev.checkout(branch="branch1")
The following files in your working directory contain uncommitted changes:

     tracked

Discard changes? [discard/Cancel/stash] cancel
Aborting checkout of branch "branch1".

#  (use "sage --dev commit" to save changes in a new commit)
```

We can stash uncommitted changes:

```
sage: UI.append("s")
sage: dev.checkout(branch="branch1")
The following files in your working directory contain uncommitted changes:

     tracked

Discard changes? [discard/Cancel/stash] s
Your changes have been moved to the git stash stack. To re-apply your changes
later use "git stash apply".
On local branch "branch1" without associated ticket.

#  Use "sage --dev merge" to include another ticket/branch.
#  Use "sage --dev commit" to save changes into a new commit.
```

And retrieve the stashed changes later:

```
sage: dev.checkout(branch='branch2')
On local branch "branch2" without associated ticket.

#  Use "sage --dev merge" to include another ticket/branch.
#  Use "sage --dev commit" to save changes into a new commit.
sage: dev.git.echo.stash('apply')
# On branch branch2
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
#   modified:   tracked
#
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#   untracked
no changes added to commit (use "git add" and/or "git commit -a")
```

Or we can just discard the changes:

```
sage: UI.append("discard")
sage: dev.checkout(branch="branch1")
The following files in your working directory contain uncommitted changes:

     tracked

Discard changes? [discard/Cancel/stash] discard
On local branch "branch1" without associated ticket.
```

```
#  Use "sage --dev merge" to include another ticket/branch.
#  Use "sage --dev commit" to save changes into a new commit.
```

Checking out a branch when in the middle of a merge:

```
sage: dev.git.super_silent.checkout('-b','merge_branch')
sage: with open('merge', 'w') as f: f.write("version 0")
sage: dev.git.silent.add('merge')
sage: dev.git.silent.commit('-m','some change')
sage: dev.git.super_silent.checkout('branch1')
sage: with open('merge', 'w') as f: f.write("version 1")
sage: dev.git.silent.add('merge')
sage: dev.git.silent.commit('-m','conflicting change')
sage: from sage.dev.git_error import GitError
sage: try:
....:     dev.git.silent.merge('merge_branch')
....: except GitError: pass
sage: UI.append('r')
sage: dev.checkout(branch='merge_branch')
Repository is in an unclean state (merge). Resetting the state will discard any
uncommited changes.
Reset repository? [reset/Cancel] r
On local branch "merge_branch" without associated ticket.

#  Use "sage --dev merge" to include another ticket/branch.
#  Use "sage --dev commit" to save changes into a new commit.
```

Checking out a branch when in a detached HEAD:

```
sage: dev.git.super_silent.checkout('branch2', detach=True)
sage: dev.checkout(branch='branch1')
On local branch "branch1" without associated ticket.

#  Use "sage --dev merge" to include another ticket/branch.
#  Use "sage --dev commit" to save changes into a new commit.
```

With uncommitted changes:

```
sage: dev.git.super_silent.checkout('branch2', detach=True)
sage: with open('tracked', 'w') as f: f.write("boo")
sage: UI.append("discard")
sage: dev.checkout(branch='branch1')
The following files in your working directory contain uncommitted changes:

    tracked

Discard changes? [discard/Cancel/stash] discard
On local branch "branch1" without associated ticket.

#  Use "sage --dev merge" to include another ticket/branch.
#  Use "sage --dev commit" to save changes into a new commit.
```

Checking out a branch with untracked files that would be overwritten by the checkout:

```
sage: with open('tracked', 'w') as f: f.write("boo")
sage: dev.checkout(branch='branch2')
GitError: git exited with a non-zero exit code (1).
This happened while executing "git -c user.email=doc@test.test -c
user.name=doctest checkout branch2".
```

```
git printed nothing to STDOUT.
git printed the following to STDERR:
error: The following untracked working tree files would be overwritten
by checkout:
    tracked
Please move or remove them before you can switch branches.
Aborting
```

checkout_ticket (*ticket*, *branch=None*, *base=''*)

Checkout the branch associated to `ticket`.

If `branch` is an existing local branch, then `ticket` will be associated to it, and `branch` will be checked out into the working directory.

Otherwise, if there is no local branch for `ticket`, the branch specified on trac will be pulled to `branch` unless `base` is set to something other than the empty string `''`. If the trac ticket does not specify a branch yet or if `base` is not the empty string, then a new one will be created from `base` (per default, the master branch).

INPUT:

- `ticket` – a string or an integer identifying a ticket
- `branch` – a string, the name of the local branch that stores changes for `ticket` (default: ticket/`ticket`)
- `base` – a string or `None`, a branch on which to base a new branch if one is going to be created (default: the empty string `''` to create the new branch from the master branch), or a ticket; if `base` is set to `None`, then the current ticket is used. If `base` is a ticket, then the corresponding dependency will be added.

See Also:

`pull()`, `create_ticket()`, `vanilla()`

TESTS:

Create a doctest setup with two users:

```
sage: from sage.dev.test.sagedev import two_user_setup
sage: alice, config_alice, bob, config_bob, server = two_user_setup()
```

Alice tries to checkout ticket #1 which does not exist yet:

```
sage: alice._chdir()
sage: alice.checkout(ticket=1)
Ticket name "1" is not valid or ticket does not exist on trac.
```

Bob creates that ticket:

```
sage: bob._chdir()
sage: bob._UI.append("Summary: summary1\ndescription")
sage: bob.create_ticket()
Created ticket #1 at https://trac.sagemath.org/1.

#  (use "sage --dev checkout --ticket=1" to create a new local branch)
1
sage: bob.checkout(ticket=1)
On ticket #1 with associated local branch "ticket/1".

#  Use "sage --dev merge" to include another ticket/branch.
#  Use "sage --dev commit" to save changes into a new commit.
```

Now alice can check it out, even though there is no branch on the ticket description:

```
sage: alice._chdir()
sage: alice.checkout(ticket=1)
On ticket #1 with associated local branch "ticket/1".

#  Use "sage --dev merge" to include another ticket/branch.
#  Use "sage --dev commit" to save changes into a new commit.
```

If Bob commits something to the ticket, a `checkout` by Alice does not take his changes into account:

```
sage: bob._chdir()
sage: bob.git.super_silent.commit(allow_empty=True,message="empty commit")
sage: bob._UI.append("y")
sage: bob.push()
The branch "u/bob/ticket/1" does not exist on the remote server.
Create new remote branch? [Yes/no] y

sage: alice._chdir()
sage: alice.checkout(ticket=1)
On ticket #1 with associated local branch "ticket/1".

#  Use "sage --dev merge" to include another ticket/branch.
#  Use "sage --dev commit" to save changes into a new commit.
sage: alice.git.echo.log('--pretty=%s')
initial commit
```

If Alice had not checked that ticket out before, she would of course see Bob's changes (this also checks that we can handle a corrupt ticket database and a detached HEAD):

```
sage: alice.git.super_silent.checkout('HEAD', detach=True)
sage: alice.git.super_silent.branch('-d','ticket/1')
sage: alice.checkout(ticket=1) # ticket #1 refers to the non-existant branch 'ticket/1'
Ticket #1 refers to the non-existant local branch "ticket/1". If you have not
manually interacted with git, then this is a bug in sagedev. Removing the
association from ticket #1 to branch "ticket/1".
On ticket #1 with associated local branch "ticket/1".

#  Use "sage --dev merge" to include another ticket/branch.
#  Use "sage --dev commit" to save changes into a new commit.
sage: alice.git.current_branch()
'ticket/1'
sage: alice.git.echo.log('--pretty=%s')
empty commit
initial commit
```

Checking out a ticket with untracked files:

```
sage: alice._UI.append("Summary: summary2\ndescription")
sage: alice.create_ticket()
Created ticket #2 at https://trac.sagemath.org/2.

#  (use "sage --dev checkout --ticket=2" to create a new local branch)
2
sage: alice.checkout(ticket=2)
On ticket #2 with associated local branch "ticket/2".

#  Use "sage --dev merge" to include another ticket/branch.
#  Use "sage --dev commit" to save changes into a new commit.
sage: alice.git.echo.log('--pretty=%s')
initial commit
```

```
sage: open("untracked","w").close()
sage: alice.checkout(ticket=1)
On ticket #1 with associated local branch "ticket/1".

#  Use "sage --dev merge" to include another ticket/branch.
#  Use "sage --dev commit" to save changes into a new commit.
sage: alice.git.echo.log('--pretty=%s')
empty commit
initial commit
```

Checking out a ticket with untracked files which make a checkout impossible:

```
sage: alice.git.super_silent.add("untracked")
sage: alice.git.super_silent.commit(message="added untracked")
sage: alice.checkout(ticket=2)
On ticket #2 with associated local branch "ticket/2".

#  Use "sage --dev merge" to include another ticket/branch.
#  Use "sage --dev commit" to save changes into a new commit.
sage: open("untracked","w").close()
sage: alice.checkout(ticket=1)
GitError: git exited with a non-zero exit code (1).
This happened while executing "git -c user.email=doc@test.test -c
user.name=alice checkout ticket/1".
git printed nothing to STDOUT.
git printed the following to STDERR:
error: The following untracked working tree files would be overwritten by checkout:
    untracked
Please move or remove them before you can switch branches.
Aborting
```

Checking out a ticket with uncommited changes:

```
sage: open("tracked", "w").close()
sage: alice.git.super_silent.add("tracked")
sage: alice._UI.append('d')
sage: alice.checkout(ticket=2)
The following files in your working directory contain uncommitted changes:

     tracked

Discard changes? [discard/Keep/stash] d
On ticket #2 with associated local branch "ticket/2".

#  Use "sage --dev merge" to include another ticket/branch.
#  Use "sage --dev commit" to save changes into a new commit.
```

Now follow some single user tests to check that the parameters are interpreted correctly:

```
sage: from sage.dev.test.sagedev import single_user_setup
sage: dev, config, UI, server = single_user_setup()
sage: dev._wrap("_dependencies_for_ticket")
```

First, create some tickets:

```
sage: UI.append("Summary: ticket1\ndescription")
sage: dev.create_ticket()
Created ticket #1 at https://trac.sagemath.org/1.

#  (use "sage --dev checkout --ticket=1" to create a new local branch)
```

```
1
sage: dev.checkout(ticket=1)
On ticket #1 with associated local branch "ticket/1".

#  Use "sage --dev merge" to include another ticket/branch.
#  Use "sage --dev commit" to save changes into a new commit.
sage: UI.append("Summary: ticket2\ndescription")
sage: dev.create_ticket()
Created ticket #2 at https://trac.sagemath.org/2.

#  (use "sage --dev checkout --ticket=2" to create a new local branch)
2
sage: dev.checkout(ticket=2)
On ticket #2 with associated local branch "ticket/2".

#  Use "sage --dev merge" to include another ticket/branch.
#  Use "sage --dev commit" to save changes into a new commit.
sage: dev.git.silent.commit(allow_empty=True, message="second commit")
sage: dev.git.commit_for_branch('ticket/2') != dev.git.commit_for_branch('ticket/1')
True
```

Check that `base` works:

```
sage: UI.append("Summary: ticket3\ndescription")
sage: dev.create_ticket()
Created ticket #3 at https://trac.sagemath.org/3.

#  (use "sage --dev checkout --ticket=3" to create a new local branch)
3
sage: dev.checkout(ticket=3, base=2)
On ticket #3 with associated local branch "ticket/3".

#  Use "sage --dev merge" to include another ticket/branch.
#  Use "sage --dev commit" to save changes into a new commit.
sage: dev.git.commit_for_branch('ticket/3') == dev.git.commit_for_branch('ticket/2')
True
sage: dev._dependencies_for_ticket(3)
(2,)
sage: UI.append("Summary: ticket4\ndescription")
sage: dev.create_ticket()
Created ticket #4 at https://trac.sagemath.org/4.

#  (use "sage --dev checkout --ticket=4" to create a new local branch)
4
sage: dev.checkout(ticket=4, base='ticket/2')
On ticket #4 with associated local branch "ticket/4".

#  Use "sage --dev merge" to include another ticket/branch.
#  Use "sage --dev commit" to save changes into a new commit.
sage: dev.git.commit_for_branch('ticket/4') == dev.git.commit_for_branch('ticket/2')
True
sage: dev._dependencies_for_ticket(4)
()
```

In this example `base` does not exist:

```
sage: UI.append("Summary: ticket5\ndescription")
sage: dev.create_ticket()
Created ticket #5 at https://trac.sagemath.org/5.
```

```
#  (use "sage --dev checkout --ticket=5" to create a new local branch)
5
sage: dev.checkout(ticket=5, base=1000)
Ticket name "1000" is not valid or ticket does not exist on trac.
```

In this example `base` does not exist locally:

```
sage: UI.append("Summary: ticket6\ndescription")
sage: dev.create_ticket()
Created ticket #6 at https://trac.sagemath.org/6.

#  (use "sage --dev checkout --ticket=6" to create a new local branch)
6
sage: dev.checkout(ticket=6, base=5)
Branch field is not set for ticket #5 on trac.
```

Creating a ticket when in detached HEAD state:

```
sage: dev.git.super_silent.checkout('HEAD', detach=True)
sage: UI.append("Summary: ticket detached\ndescription")
sage: dev.create_ticket()
Created ticket #7 at https://trac.sagemath.org/7.

#  (use "sage --dev checkout --ticket=7" to create a new local branch)
7
sage: dev.checkout(ticket=7)
On ticket #7 with associated local branch "ticket/7".

#  Use "sage --dev merge" to include another ticket/branch.
#  Use "sage --dev commit" to save changes into a new commit.
sage: dev.git.current_branch()
'ticket/7'
```

Creating a ticket when in the middle of a merge:

```
sage: dev.git.super_silent.checkout('-b','merge_branch')
sage: with open('merge', 'w') as f: f.write("version 0")
sage: dev.git.silent.add('merge')
sage: dev.git.silent.commit('-m','some change')
sage: dev.git.super_silent.checkout('ticket/7')
sage: with open('merge', 'w') as f: f.write("version 1")
sage: dev.git.silent.add('merge')
sage: dev.git.silent.commit('-m','conflicting change')
sage: from sage.dev.git_error import GitError
sage: try:
....:     dev.git.silent.merge('merge_branch')
....: except GitError: pass
sage: UI.append("Summary: ticket merge\ndescription")
sage: dev.create_ticket()
Created ticket #8 at https://trac.sagemath.org/8.

#  (use "sage --dev checkout --ticket=8" to create a new local branch)
8
sage: UI.append("cancel")
sage: dev.checkout(ticket=8)
Repository is in an unclean state (merge). Resetting the state will discard any
uncommited changes.
Reset repository? [reset/Cancel] cancel
Aborting checkout of branch "ticket/8".
```

```
#  (use "sage --dev commit" to save changes in a new commit)
sage: dev.git.reset_to_clean_state()
```

Creating a ticket with uncommitted changes:

```
sage: open('tracked', 'w').close()
sage: dev.git.silent.add('tracked')
sage: UI.append("Summary: ticket merge\ndescription")
sage: dev.create_ticket()
Created ticket #9 at https://trac.sagemath.org/9.

#  (use "sage --dev checkout --ticket=9" to create a new local branch)
9
```

The new branch is based on master which is not the same commit as the current branch `ticket/7`, so it is not a valid option to `'keep'` changes:

```
sage: UI.append("cancel")
sage: dev.checkout(ticket=9)
The following files in your working directory contain uncommitted changes:

    tracked

Discard changes? [discard/Cancel/stash] cancel
Aborting checkout of branch "ticket/9".

#  (use "sage --dev commit" to save changes in a new commit)
```

Finally, in this case we can keep changes because the base is the same commit as the current branch:

```
sage: UI.append("Summary: ticket merge\ndescription")
sage: dev.create_ticket()
Created ticket #10 at https://trac.sagemath.org/10.

#  (use "sage --dev checkout --ticket=10" to create a new local branch)
10
sage: UI.append("keep")
sage: dev.checkout(ticket=10, base='ticket/7')
The following files in your working directory contain uncommitted changes:

    tracked

Discard changes? [discard/Keep/stash] keep
On ticket #10 with associated local branch "ticket/10".

#  Use "sage --dev merge" to include another ticket/branch.
#  Use "sage --dev commit" to save changes into a new commit.
```

clean(*error_unless_clean=True*)

Restore the working directory to the most recent commit.

INPUT:

- `error_unless_clean` – a boolean (default: `True`), whether to raise an `user_interface_error.OperationCancelledError` if the directory remains in an unclean state; used internally.

TESTS:

Set up a single user for doctesting:

```
sage: from sage.dev.test.sagedev import single_user_setup
sage: dev, config, UI, server = single_user_setup()
```

Check that nothing happens if there no changes:

```
sage: dev.clean()
```

Check that nothing happens if there are only untracked files:

```
sage: open("untracked","w").close()
sage: dev.clean()
```

Uncommitted changes can simply be dropped:

```
sage: open("tracked","w").close()
sage: dev.git.silent.add("tracked")
sage: dev.git.silent.commit(message="added tracked")
sage: with open("tracked", "w") as f: f.write("foo")
sage: UI.append("discard")
sage: dev.clean()
The following files in your working directory contain uncommitted changes:

     tracked

Discard changes? [discard/Cancel/stash] discard
sage: dev.clean()
```

Uncommitted changes can be kept:

```
sage: with open("tracked", "w") as f: f.write("foo")
sage: UI.append("cancel")
sage: dev.clean()
The following files in your working directory contain uncommitted changes:

     tracked

Discard changes? [discard/Cancel/stash] cancel
```

Or stashed:

```
sage: UI.append("stash")
sage: dev.clean()
The following files in your working directory contain uncommitted changes:

     tracked

Discard changes? [discard/Cancel/stash] stash
Your changes have been moved to the git stash stack. To re-apply your changes
later use "git stash apply".
sage: dev.clean()
```

comment (*ticket=None*)

Add a comment to `ticket` on trac.

INPUT:

- `ticket` – an integer or string identifying a ticket or `None` (default: `None`), the number of the ticket to edit. If `None`, edit the `_current_ticket()`.

See Also:

create_ticket(), edit_ticket()

TESTS:

Set up a single user for doctesting:

```
sage: from sage.dev.test.sagedev import single_user_setup
sage: dev, config, UI, server = single_user_setup()
```

Create a ticket and add a comment:

```
sage: UI.append("Summary: summary1\ndescription")
sage: dev.create_ticket()
Created ticket #1 at https://trac.sagemath.org/1.

#  (use "sage --dev checkout --ticket=1" to create a new local branch)
1
sage: dev.checkout(ticket=1)
On ticket #1 with associated local branch "ticket/1".

#  Use "sage --dev merge" to include another ticket/branch.
#  Use "sage --dev commit" to save changes into a new commit.
sage: UI.append("comment")
sage: dev.comment()
sage: server.tickets[1].comments
['comment']
```

commit (*message=None*, *interactive=False*)

Create a commit from the pending changes on the current branch.

This is most akin to mercurial's commit command, not git's, since we do not require users to add files.

INPUT:

- `message` – the message of the commit (default: `None`), if `None`, prompt for a message.
- `interactive` – if set, interactively select which part of the changes should be part of the commit

See Also:

- `push()` – Push changes to the remote server. This is the next step once you've committed some changes.
- `diff()` – Show changes that will be committed.

TESTS:

Set up a single user for doctesting:

```
sage: from sage.dev.test.sagedev import single_user_setup
sage: dev, config, UI, server = single_user_setup()
```

Commit an untracked file:

```
sage: dev.git.super_silent.checkout('-b', 'branch1')
sage: open("tracked","w").close()
sage: dev._UI.extend(["y", "added tracked", "y", "y"])
sage: dev.commit()
The following files in your working directory are not tracked by git:

    tracked

Start tracking any of these files? [yes/No] y
```

```
Start tracking "tracked"? [yes/No] y
Commit your changes to branch "branch1"? [Yes/no] y

#  Use "sage --dev push" to push your commits to the trac server once you are done.
```

Commit a tracked file:

```
sage: with open("tracked", "w") as F: F.write("foo")
sage: dev._UI.append('y')
sage: dev.commit(message='modified tracked')
Commit your changes to branch "branch1"? [Yes/no] y

#  Use "sage --dev push" to push your commits to the trac server once you are done.
```

create_ticket()

Create a new ticket on trac.

OUTPUT:

Returns the number of the newly created ticket as an int.

See Also:

checkout(), pull(), edit_ticket()

TESTS:

Set up a single user environment:

```
sage: from sage.dev.test.sagedev import single_user_setup
sage: dev, config, UI, server = single_user_setup()
sage: dev._wrap("_dependencies_for_ticket")
```

Create some tickets:

```
sage: UI.append("Summary: ticket1\ndescription")
sage: dev.create_ticket()
Created ticket #1 at https://trac.sagemath.org/1.

#  (use "sage --dev checkout --ticket=1" to create a new local branch)
1

sage: UI.append("Summary: ticket2\ndescription")
sage: dev.create_ticket()
Created ticket #2 at https://trac.sagemath.org/2.

#  (use "sage --dev checkout --ticket=2" to create a new local branch)
2
```

This fails if the internet connection is broken:

```
sage: dev.trac._connected = False
sage: UI.append("Summary: ticket7\ndescription")
sage: dev.create_ticket()
A network error ocurred, ticket creation aborted.
Your command failed because no connection to trac could be established.
sage: dev.trac._connected = True
```

diff(*base='commit'*)

Show how the current file system differs from `base`.

INPUT:

- `base` – a string; show the differences against the latest `'commit'` (the default), against the branch `'master'` (or any other branch name), or the merge of the `'dependencies'` of the current ticket (if the dependencies merge cleanly)

See Also:

- `commit()` – record changes into the repository.
- `tickets()` – list local tickets (you may want to commit your changes to a branch other than the current one).

TESTS:

Create a doctest setup with a single user:

```
sage: from sage.dev.test.sagedev import single_user_setup
sage: dev, config, UI, server = single_user_setup()
```

Create some tickets and make one depend on the others:

```
sage: UI.append("Summary: summary\ndescription")
sage: dev.create_ticket()
Created ticket #1 at https://trac.sagemath.org/1.

#  (use "sage --dev checkout --ticket=1" to create a new local branch)
1
sage: dev.checkout(ticket=1)
On ticket #1 with associated local branch "ticket/1".

#  Use "sage --dev merge" to include another ticket/branch.
#  Use "sage --dev commit" to save changes into a new commit.
sage: UI.append("y")
sage: dev.push()
The branch "u/doctest/ticket/1" does not exist on the remote server.
Create new remote branch? [Yes/no] y
sage: UI.append("Summary: summary\ndescription")
sage: dev.create_ticket()
Created ticket #2 at https://trac.sagemath.org/2.

#  (use "sage --dev checkout --ticket=2" to create a new local branch)
2
sage: dev.checkout(ticket=2)
On ticket #2 with associated local branch "ticket/2".

#  Use "sage --dev merge" to include another ticket/branch.
#  Use "sage --dev commit" to save changes into a new commit.
sage: UI.append("y")
sage: dev.push()
The branch "u/doctest/ticket/2" does not exist on the remote server.
Create new remote branch? [Yes/no] y
sage: UI.append("Summary: summary\ndescription")
sage: dev.create_ticket()
Created ticket #3 at https://trac.sagemath.org/3.

#  (use "sage --dev checkout --ticket=3" to create a new local branch)
3
sage: dev.checkout(ticket=3)
On ticket #3 with associated local branch "ticket/3".

#  Use "sage --dev merge" to include another ticket/branch.
```

```
#  Use "sage --dev commit" to save changes into a new commit.
sage: UI.append("y")
sage: dev.push()
The branch "u/doctest/ticket/3" does not exist on the remote server.
Create new remote branch? [Yes/no] y
sage: dev.merge("#1")
Merging the remote branch "u/doctest/ticket/1" into the local branch "ticket/3".
Automatic merge successful.

#  (use "sage --dev commit" to commit your merge)

Added dependency on #1 to #3.
sage: dev.merge("#2")
Merging the remote branch "u/doctest/ticket/2" into the local branch "ticket/3".
Automatic merge successful.

#  (use "sage --dev commit" to commit your merge)

Added dependency on #2 to #3.
```

Make some non-conflicting changes on the tickets:

```
sage: dev.checkout(ticket="#1")
On ticket #1 with associated local branch "ticket/1".

#  Use "sage --dev merge" to include another ticket/branch.
#  Use "sage --dev commit" to save changes into a new commit.
sage: with open("ticket1","w") as f: f.write("ticket1")
sage: dev.git.silent.add("ticket1")
sage: dev.git.super_silent.commit(message="added ticket1")

sage: dev.checkout(ticket="#2")
On ticket #2 with associated local branch "ticket/2".

#  Use "sage --dev merge" to include another ticket/branch.
#  Use "sage --dev commit" to save changes into a new commit.
sage: with open("ticket2","w") as f: f.write("ticket2")
sage: dev.git.silent.add("ticket2")
sage: dev.git.super_silent.commit(message="added ticket2")
sage: UI.append("y")
sage: dev.push()
Local commits that are not on the remote branch "u/doctest/ticket/2":

    ...: added ticket2

Push to remote branch? [Yes/no] y

sage: dev.checkout(ticket="#3")
On ticket #3 with associated local branch "ticket/3".

#  Use "sage --dev merge" to include another ticket/branch.
#  Use "sage --dev commit" to save changes into a new commit.
sage: open("ticket3","w").close()
sage: dev.git.silent.add("ticket3")
sage: dev.git.super_silent.commit(message="added ticket3")
sage: UI.append("y")
sage: dev.push()
Local commits that are not on the remote branch "u/doctest/ticket/3":
```

```
    ...: added ticket3

Push to remote branch? [Yes/no] y
Uploading your dependencies for ticket #3: "" => "#1, #2"
```

A diff against the previous commit:

```
sage: dev.diff()
```

A diff against a ticket will always take the branch on trac:

```
sage: dev.diff("#1")
diff --git a/ticket3 b/ticket3
new file mode ...
index ...
sage: dev.diff("ticket/1")
diff --git a/ticket1 b/ticket1
deleted file mode ...
index ...
diff --git a/ticket3 b/ticket3
new file mode ...
index ...
sage: dev.checkout(ticket="#1")
On ticket #1 with associated local branch "ticket/1".

#  Use "sage --dev merge" to include another ticket/branch.
#  Use "sage --dev commit" to save changes into a new commit.
sage: UI.append("y")
sage: dev.push()
Local commits that are not on the remote branch "u/doctest/ticket/1":

    ...: added ticket1

Push to remote branch? [Yes/no] y
sage: dev.checkout(ticket="#3")
On ticket #3 with associated local branch "ticket/3".

#  Use "sage --dev merge" to include another ticket/branch.
#  Use "sage --dev commit" to save changes into a new commit.
sage: dev.diff("#1")
diff --git a/ticket1 b/ticket1
deleted file mode ...
index ...
diff --git a/ticket3 b/ticket3
new file mode ...
index ...
```

A diff against the dependencies:

```
sage: dev.diff("dependencies")
Dependency #1 has not been merged into "ticket/3" (at least not its latest
version).
#  (use "sage --dev merge --ticket=1" to merge it)

Dependency #2 has not been merged into "ticket/3" (at least not its latest
version).
#  (use "sage --dev merge --ticket=2" to merge it)

diff --git a/ticket1 b/ticket1
```

```
deleted file mode ...
index ...
diff --git a/ticket2 b/ticket2
deleted file mode ...
index ...
diff --git a/ticket3 b/ticket3
new file mode ...
index ...
sage: dev.merge("#1")
Merging the remote branch "u/doctest/ticket/1" into the local branch "ticket/3".
Automatic merge successful.

#  (use "sage --dev commit" to commit your merge)
sage: dev.merge("#2")
Merging the remote branch "u/doctest/ticket/2" into the local branch "ticket/3".
Automatic merge successful.

#  (use "sage --dev commit" to commit your merge)
sage: dev.diff("dependencies")
diff --git a/ticket3 b/ticket3
new file mode ...
index ...
```

This does not work if the dependencies do not merge:

```
sage: dev.checkout(ticket="#1")
On ticket #1 with associated local branch "ticket/1".

#  Use "sage --dev merge" to include another ticket/branch.
#  Use "sage --dev commit" to save changes into a new commit.
sage: with open("ticket2","w") as f: f.write("foo")
sage: dev.git.silent.add("ticket2")
sage: dev.git.super_silent.commit(message="added ticket2")
sage: UI.append("y")
sage: dev.push()
Local commits that are not on the remote branch "u/doctest/ticket/1":

    ...: added ticket2

Push to remote branch? [Yes/no] y

sage: dev.checkout(ticket="#3")
On ticket #3 with associated local branch "ticket/3".

#  Use "sage --dev merge" to include another ticket/branch.
#  Use "sage --dev commit" to save changes into a new commit.
sage: dev.diff("dependencies")
Dependency #1 has not been merged into "ticket/3" (at least not its latest
version).
#  (use "sage --dev merge --ticket=1" to merge it)

Dependency #2 does not merge cleanly with the other dependencies. Your diff
could not be computed.
```

edit_ticket (*ticket=None*)

Edit the description of `ticket` on trac.

INPUT:

- `ticket` – an integer or string identifying a ticket or `None` (default: `None`), the number of the ticket to edit. If `None`, edit the `_current_ticket()`.

See Also:

`create_ticket()`, `comment()`, `set_needs_review()`, `set_needs_work()`, `set_positive_review()`, `set_needs_info()`

TESTS:

Set up a single user for doctesting:

```
sage: from sage.dev.test.sagedev import single_user_setup
sage: dev, config, UI, server = single_user_setup()
```

Create a ticket and edit it:

```
sage: UI.append("Summary: summary1\ndescription")
sage: dev.create_ticket()
Created ticket #1 at https://trac.sagemath.org/1.

#  (use "sage --dev checkout --ticket=1" to create a new local branch)
1
sage: dev.checkout(ticket=1)
On ticket #1 with associated local branch "ticket/1".

#  Use "sage --dev merge" to include another ticket/branch.
#  Use "sage --dev commit" to save changes into a new commit.
sage: UI.append("Summary: summary1\ndescription...")
sage: dev.edit_ticket()
sage: dev.trac._get_attributes(1)
{'description': 'description...', 'summary': 'summary1'}
```

gather(*branch*, **tickets_or_branches*)

Create a new branch `branch` with `tickets_or_remote_branches` applied.

This method is not wrapped in the commandline dev scripts. It does nothing that cannot be done with `checkout` and `merge`, it just steepens the learning curve by introducing yet another command. Unless a clear use case emerges, it should be removed.

INPUT:

- `branch` – a string, the name of the new branch
- `tickets_or_branches` – a list of integers and strings; for an integer or string identifying a ticket, the branch on the trac ticket gets merged, for the name of a local or remote branch, that branch gets merged.

See Also:

- `merge()` – merge into the current branch rather than creating a new one

TESTS:

Create a doctest setup with a single user:

```
sage: from sage.dev.test.sagedev import single_user_setup
sage: dev, config, UI, server = single_user_setup()
```

Create tickets and branches:

```
sage: dev._UI.append("Summary: summary1\ndescription")
sage: dev.create_ticket()
Created ticket #1 at https://trac.sagemath.org/1.

#  (use "sage --dev checkout --ticket=1" to create a new local branch)
1
sage: dev.checkout(ticket=1)
On ticket #1 with associated local branch "ticket/1".

#  Use "sage --dev merge" to include another ticket/branch.
#  Use "sage --dev commit" to save changes into a new commit.
sage: open("tracked","w").close()
sage: dev.git.silent.add("tracked")
sage: dev.git.super_silent.commit(message="added tracked")
sage: dev._UI.append("y")
sage: dev._UI.append("y")
sage: dev.push()
The branch "u/doctest/ticket/1" does not exist on the remote server.
Create new remote branch? [Yes/no] y
```

Gather all these branches:

```
sage: dev._sagedev.gather("gather_branch", "#1", "ticket/1", "u/doctest/ticket/1")
```

merge(*ticket_or_branch='master'*, *pull=None*, *create_dependency=None*)

Merge changes from `ticket_or_branch` into the current branch.

Incorporate commits from other tickets/branches into the current branch.

Optionally, you can add the merged ticket to the trac "Dependency:" field. Note that the merged commits become part of the current branch, regardless of whether they are noted on trac. Adding a dependency implies the following:

- the other ticket must be positively reviewed and merged before this ticket may be merged into the official release of sage. The commits included from a dependency don't need to be reviewed in this ticket, whereas commits reviewed in this ticket from a non-dependency may make reviewing the other ticket easier.
- you can more easily merge in future changes to dependencies. So if you need a feature from another ticket it may be appropriate to create a dependency to that you may more easily benefit from others' work on that ticket.
- if you depend on another ticket then you need to worry about the progress on that ticket. If that ticket is still being actively developed then you may need to make further merges in the future if conflicts arise.

INPUT:

- `ticket_or_branch` – an integer or strings (default: `'master'`); for an integer or string identifying a ticket, the branch on the trac ticket gets merged (or the local branch for the ticket, if `pull` is `False`), for the name of a local or remote branch, that branch gets merged. If `'dependencies'`, the dependencies are merged in one by one.
- `pull` – a boolean or `None` (default: `None`); if `ticket_or_branch` identifies a ticket, whether to pull the latest branch on the trac ticket (the default); if `ticket_or_branch` is a branch name, then `pull` controls whether it should be interpreted as a remote branch (`True`) or as a local branch (`False`). If it is set to `None`, then it will take `ticket_or_branch` as a remote branch if it exists, and as a local branch otherwise.

- `create_dependency` – a boolean or `None` (default: `None`), whether to create a dependency to `ticket_or_branch`. If `None`, then a dependency is created if `ticket_or_branch` identifies a ticket and if the current branch is associated to a ticket.

Note: Dependencies are stored locally and only updated with respect to the remote server during `push()` and `pull()`.

See Also:

- `show_dependencies()` – see the current dependencies.
- `GitInterface.merge()` – git's merge command has more options and can merge multiple branches at once.

TESTS:

Create a doctest setup with two users:

```
sage: from sage.dev.test.sagedev import two_user_setup
sage: alice, config_alice, bob, config_bob, server = two_user_setup()
```

Create tickets and branches:

```
sage: alice._chdir()
sage: alice._UI.append("Summary: summary1\ndescription")
sage: alice.create_ticket()
Created ticket #1 at https://trac.sagemath.org/1.

#  (use "sage --dev checkout --ticket=1" to create a new local branch)
1
sage: alice._UI.append("Summary: summary2\ndescription")
sage: alice.create_ticket()
Created ticket #2 at https://trac.sagemath.org/2.

#  (use "sage --dev checkout --ticket=2" to create a new local branch)
2
```

Alice creates two branches and merges them:

```
sage: alice.checkout(ticket=1)
On ticket #1 with associated local branch "ticket/1".

#  Use "sage --dev merge" to include another ticket/branch.
#  Use "sage --dev commit" to save changes into a new commit.
sage: open("alice1","w").close()
sage: alice.git.silent.add("alice1")
sage: alice.git.super_silent.commit(message="added alice1")
sage: alice.checkout(ticket=2)
On ticket #2 with associated local branch "ticket/2".

#  Use "sage --dev merge" to include another ticket/branch.
#  Use "sage --dev commit" to save changes into a new commit.
sage: with open("alice2","w") as f: f.write("alice")
sage: alice.git.silent.add("alice2")
sage: alice.git.super_silent.commit(message="added alice2")
```

When merging for a ticket, the branch on the trac ticket matters:

```
sage: alice.merge("#1")
Cannot merge remote branch for #1 because no branch has been set on the trac
ticket.
sage: alice.checkout(ticket=1)
On ticket #1 with associated local branch "ticket/1".

#  Use "sage --dev merge" to include another ticket/branch.
#  Use "sage --dev commit" to save changes into a new commit.
sage: alice._UI.append("y")
sage: alice.push()
The branch "u/alice/ticket/1" does not exist on the remote server.
Create new remote branch? [Yes/no] y
sage: alice.checkout(ticket=2)
On ticket #2 with associated local branch "ticket/2".

#  Use "sage --dev merge" to include another ticket/branch.
#  Use "sage --dev commit" to save changes into a new commit.
sage: alice.merge("#1", pull=False)
Merging the local branch "ticket/1" into the local branch "ticket/2".
Automatic merge successful.

#  (use "sage --dev commit" to commit your merge)

Added dependency on #1 to #2.
```

Check that merging dependencies works:

```
sage: alice.merge("dependencies")
Merging the remote branch "u/alice/ticket/1" into the local branch "ticket/2".
Automatic merge successful.

#  (use "sage --dev commit" to commit your merge)
```

Merging local branches:

```
sage: alice.merge("ticket/1")
Merging the local branch "ticket/1" into the local branch "ticket/2".
Automatic merge successful.

#  (use "sage --dev commit" to commit your merge)
```

A remote branch for a local branch is only merged in if `pull` is set:

```
sage: alice._sagedev._set_remote_branch_for_branch("ticket/1", "nonexistant")
sage: alice.merge("ticket/1")
Merging the local branch "ticket/1" into the local branch "ticket/2".
Automatic merge successful.

#  (use "sage --dev commit" to commit your merge)
sage: alice.merge("ticket/1", pull=True)
Branch "ticket/1" does not exist on the remote system.
```

Bob creates a conflicting commit:

```
sage: bob._chdir()
sage: bob.checkout(ticket=1)
On ticket #1 with associated local branch "ticket/1".

#  Use "sage --dev merge" to include another ticket/branch.
#  Use "sage --dev commit" to save changes into a new commit.
```

```
sage: with open("alice2","w") as f: f.write("bob")
sage: bob.git.silent.add("alice2")
sage: bob.git.super_silent.commit(message="added alice2")
sage: bob._UI.append("y")
sage: bob._UI.append("y")
sage: bob.push()
The branch "u/bob/ticket/1" does not exist on the remote server.
Create new remote branch? [Yes/no] y
The branch field of ticket #1 needs to be updated from its current value
"u/alice/ticket/1" to "u/bob/ticket/1"
Change the "Branch:" field? [Yes/no] y
```

The merge now requires manual conflict resolution:

```
sage: alice._chdir()
sage: alice._UI.append("abort")
sage: alice.merge("#1")
Merging the remote branch "u/bob/ticket/1" into the local branch "ticket/2".
Automatic merge failed, there are conflicting commits.

Auto-merging alice2
CONFLICT (add/add): Merge conflict in alice2

Please edit the affected files to resolve the conflicts. When you are finished,
your resolution will be commited.
Finished? [ok/Abort] abort
sage: alice._UI.append("ok")
sage: alice.merge("#1")
Merging the remote branch "u/bob/ticket/1" into the local branch "ticket/2".
Automatic merge failed, there are conflicting commits.

Auto-merging alice2
CONFLICT (add/add): Merge conflict in alice2

Please edit the affected files to resolve the conflicts. When you are finished,
your resolution will be commited.
Finished? [ok/Abort] ok
Created a commit from your conflict resolution.
```

We cannot merge a ticket into itself:

```
sage: alice.merge(2)
cannot merge a ticket into itself
```

We also cannot merge if the working directory has uncommited changes:

```
sage: alice._UI.append("cancel")
sage: with open("alice2","w") as f: f.write("uncommited change")
sage: alice.merge(1)
The following files in your working directory contain uncommitted changes:

    alice2

Discard changes? [discard/Cancel/stash] cancel
Cannot merge because working directory is not in a clean state.

#  (use "sage --dev commit" to commit your changes)
```

needs_info(*ticket=None*, *comment=''*)

Set a ticket on trac to `needs_info`.

INPUT:

- `ticket` – an integer or string identifying a ticket or `None` (default: `None`), the number of the ticket to edit. If `None`, edit the `_current_ticket()`.
- `comment` – a comment to go with the status change.

See Also:

`edit_ticket()`, `needs_review()`, `positive_review()`, `comment()`, `needs_work()`

TESTS:

Create a doctest setup with two users:

```
sage: from sage.dev.test.sagedev import two_user_setup
sage: alice, config_alice, bob, config_bob, server = two_user_setup()
```

Alice creates a ticket and set it to needs_review:

```
sage: alice._chdir()
sage: alice._UI.append("Summary: summary1\ndescription")
sage: alice.create_ticket()
Created ticket #1 at https://trac.sagemath.org/1.

#  (use "sage --dev checkout --ticket=1" to create a new local branch)
1
sage: alice.checkout(ticket=1)
On ticket #1 with associated local branch "ticket/1".

#  Use "sage --dev merge" to include another ticket/branch.
#  Use "sage --dev commit" to save changes into a new commit.
sage: open("tracked", "w").close()
sage: alice.git.super_silent.add("tracked")
sage: alice.git.super_silent.commit(message="alice: added tracked")
sage: alice._UI.append("y")
sage: alice.push()
The branch "u/alice/ticket/1" does not exist on the remote server.
Create new remote branch? [Yes/no] y
sage: alice.needs_review(comment='Review my ticket!')
```

Bob reviews the ticket and finds it lacking:

```
sage: bob._chdir()
sage: bob.checkout(ticket=1)
On ticket #1 with associated local branch "ticket/1".

#  Use "sage --dev merge" to include another ticket/branch.
#  Use "sage --dev commit" to save changes into a new commit.
sage: bob.needs_info(comment='Why is a tracked file enough?')
sage: bob.trac._get_attributes(1)['status']
'needs_info'
```

needs_review(*ticket=None*, *comment=''*)

Set a ticket on trac to `needs_review`.

INPUT:

- `ticket` – an integer or string identifying a ticket or `None` (default: `None`), the number of the ticket to edit. If `None`, edit the `_current_ticket()`.

- `comment` – a comment to go with the status change.

See Also:

`edit_ticket()`, `set_needs_work()`, `set_positive_review()`, `comment()`, `set_needs_info()`

TESTS:

Set up a single user for doctesting:

```
sage: from sage.dev.test.sagedev import single_user_setup
sage: dev, config, UI, server = single_user_setup()
```

Create a ticket and set it to needs_review:

```
sage: UI.append("Summary: summary1\ndescription")
sage: dev.create_ticket()
Created ticket #1 at https://trac.sagemath.org/1.

#  (use "sage --dev checkout --ticket=1" to create a new local branch)
1
sage: dev.checkout(ticket=1)
On ticket #1 with associated local branch "ticket/1".

#  Use "sage --dev merge" to include another ticket/branch.
#  Use "sage --dev commit" to save changes into a new commit.
sage: open("tracked", "w").close()
sage: dev.git.super_silent.add("tracked")
sage: dev.git.super_silent.commit(message="alice: added tracked")
sage: dev._UI.append("y")
sage: dev.push()
The branch "u/doctest/ticket/1" does not exist on the remote server.
Create new remote branch? [Yes/no] y
sage: dev.needs_review(comment='Review my ticket!')
sage: dev.trac._get_attributes(1)['status']
'needs_review'
```

needs_work(*ticket=None*, *comment=''*)

Set a ticket on trac to `needs_work`.

INPUT:

- `ticket` – an integer or string identifying a ticket or `None` (default: `None`), the number of the ticket to edit. If `None`, edit the `_current_ticket()`.
- `comment` – a comment to go with the status change.

See Also:

`edit_ticket()`, `set_needs_review()`, `set_positive_review()`, `comment()`, `set_needs_info()`

TESTS:

Create a doctest setup with two users:

```
sage: from sage.dev.test.sagedev import two_user_setup
sage: alice, config_alice, bob, config_bob, server = two_user_setup()
```

Alice creates a ticket and set it to needs_review:

```
sage: alice._chdir()
sage: alice._UI.append("Summary: summary1\ndescription")
```

```
sage: alice.create_ticket()
Created ticket #1 at https://trac.sagemath.org/1.

#  (use "sage --dev checkout --ticket=1" to create a new local branch)
1
sage: alice.checkout(ticket=1)
On ticket #1 with associated local branch "ticket/1".

#  Use "sage --dev merge" to include another ticket/branch.
#  Use "sage --dev commit" to save changes into a new commit.
sage: open("tracked", "w").close()
sage: alice.git.super_silent.add("tracked")
sage: alice.git.super_silent.commit(message="alice: added tracked")
sage: alice._UI.append("y")
sage: alice.push()
The branch "u/alice/ticket/1" does not exist on the remote server.
Create new remote branch? [Yes/no] y
sage: alice.needs_review(comment='Review my ticket!')
```

Bob reviews the ticket and finds it lacking:

```
sage: bob._chdir()
sage: bob.checkout(ticket=1)
On ticket #1 with associated local branch "ticket/1".

#  Use "sage --dev merge" to include another ticket/branch.
#  Use "sage --dev commit" to save changes into a new commit.
sage: bob.needs_work(comment='Need to add an untracked file!')
sage: bob.trac._get_attributes(1)['status']
'needs_work'
```

positive_review (*ticket=None*, *comment=''*)

Set a ticket on trac to `positive_review`.

INPUT:

- `ticket` – an integer or string identifying a ticket or `None` (default: `None`), the number of the ticket to edit. If `None`, edit the `_current_ticket()`.
- `comment` – a comment to go with the status change.

See Also:

`edit_ticket()`, `needs_review()`, `needs_info()`, `comment()`, `needs_work()`

TESTS:

Create a doctest setup with two users:

```
sage: from sage.dev.test.sagedev import two_user_setup
sage: alice, config_alice, bob, config_bob, server = two_user_setup()
```

Alice creates a ticket and set it to needs_review:

```
sage: alice._chdir()
sage: alice._UI.append("Summary: summary1\ndescription")
sage: alice.create_ticket()
Created ticket #1 at https://trac.sagemath.org/1.

#  (use "sage --dev checkout --ticket=1" to create a new local branch)
1
```

```
sage: alice.checkout(ticket=1)
On ticket #1 with associated local branch "ticket/1".

#  Use "sage --dev merge" to include another ticket/branch.
#  Use "sage --dev commit" to save changes into a new commit.
sage: open("tracked", "w").close()
sage: alice.git.super_silent.add("tracked")
sage: alice.git.super_silent.commit(message="alice: added tracked")
sage: alice._UI.append("y")
sage: alice.push()
The branch "u/alice/ticket/1" does not exist on the remote server.
Create new remote branch? [Yes/no] y
sage: alice.needs_review(comment='Review my ticket!')
```

Bob reviews the ticket and finds it good:

```
sage: bob._chdir()
sage: bob.checkout(ticket=1)
On ticket #1 with associated local branch "ticket/1".

#  Use "sage --dev merge" to include another ticket/branch.
#  Use "sage --dev commit" to save changes into a new commit.
sage: bob.positive_review()
sage: bob.trac._get_attributes(1)['status']
'positive_review'
```

prune_tickets()

Remove branches for tickets that are already merged into master.

See Also:

abandon() – Abandon a single ticket or branch.

TESTS:

Create a single user for doctesting:

```
sage: from sage.dev.test.sagedev import single_user_setup
sage: dev, config, UI, server = single_user_setup()
```

Create a ticket branch:

```
sage: UI.append("Summary: summary\ndescription")
sage: dev.create_ticket()
Created ticket #1 at https://trac.sagemath.org/1.

#  (use "sage --dev checkout --ticket=1" to create a new local branch)
1
sage: dev.checkout(ticket=1)
On ticket #1 with associated local branch "ticket/1".

#  Use "sage --dev merge" to include another ticket/branch.
#  Use "sage --dev commit" to save changes into a new commit.
sage: dev.tickets()
    : master
* #1: ticket/1 summary
```

With a commit on it, the branch is not abandoned:

```
sage: open("tracked","w").close()
sage: dev.git.silent.add("tracked")
```

```
sage: dev.git.super_silent.commit(message="added tracked")
sage: dev.prune_tickets()
sage: dev.tickets()
    : master
* #1: ticket/1 summary
```

After merging it to the master branch, it is abandoned. This does not work, because we cannot move the current branch:

```
sage: dev.git.super_silent.checkout("master")
sage: dev.git.super_silent.merge("ticket/1")

sage: dev.git.super_silent.checkout("ticket/1")
sage: dev.prune_tickets()
Abandoning #1.
Cannot delete "ticket/1": is the current branch.

#  (use "sage --dev vanilla" to switch to the master branch)
```

Now, the branch is abandoned:

```
sage: dev.vanilla()
sage: dev.prune_tickets()
Abandoning #1.
Moved your branch "ticket/1" to "trash/ticket/1".
sage: dev.tickets()
: master
sage: dev.prune_tickets()
```

pull(*ticket_or_remote_branch=None*)

Pull `ticket_or_remote_branch` to `branch`.

INPUT:

- `ticket_or_remote_branch` – a string or an integer or `None` (default: `None`), a ticket or a remote branch name; setting this to `None` has the same effect as setting it to the `current_ticket()`.

TESTS:

Create a doctest setup with two users:

```
sage: from sage.dev.test.sagedev import two_user_setup
sage: alice, config_alice, bob, config_bob, server = two_user_setup()
```

Alice creates ticket 1:

```
sage: alice._chdir()
sage: alice._UI.append("Summary: summary1\ndescription")
sage: alice.create_ticket()
Created ticket #1 at https://trac.sagemath.org/1.

#  (use "sage --dev checkout --ticket=1" to create a new local branch)
1
sage: alice.checkout(ticket=1)
On ticket #1 with associated local branch "ticket/1".

#  Use "sage --dev merge" to include another ticket/branch.
#  Use "sage --dev commit" to save changes into a new commit.
```

Bob attempts to pull for the ticket but fails because there is no branch for the ticket yet:

```
sage: bob._chdir()
sage: bob.pull(1)
Branch field is not set for ticket #1 on trac.
```

So, Bob starts to work on the ticket on a new branch:

```
sage: bob.checkout(ticket=1)
On ticket #1 with associated local branch "ticket/1".

#  Use "sage --dev merge" to include another ticket/branch.
#  Use "sage --dev commit" to save changes into a new commit.
```

Alice pushes a commit:

```
sage: alice._chdir()
sage: alice.git.super_silent.commit(allow_empty=True, message="alice: empty commit")
sage: alice._UI.append("y")
sage: alice.push()
The branch "u/alice/ticket/1" does not exist on the remote server.
Create new remote branch? [Yes/no] y
```

Bob pulls the changes for ticket 1:

```
sage: bob._chdir()
sage: bob.pull()
Merging the remote branch "u/alice/ticket/1" into the local branch "ticket/1".
Automatic merge successful.

#  (use "sage --dev commit" to commit your merge)
sage: bob.git.echo.log('--pretty=%s')
alice: empty commit
initial commit
```

Bob commits a change:

```
sage: open("bobs_file","w").close()
sage: bob.git.silent.add("bobs_file")
sage: bob.git.super_silent.commit(message="bob: added bobs_file")
sage: bob._UI.append("y")
sage: bob._UI.append("y")
sage: bob.push()
The branch "u/bob/ticket/1" does not exist on the remote server.
Create new remote branch? [Yes/no] y
The branch field of ticket #1 needs to be updated from its current value
"u/alice/ticket/1" to "u/bob/ticket/1"
Change the "Branch:" field? [Yes/no] y
```

Alice commits non-conflicting changes:

```
sage: alice._chdir()
sage: with open("alices_file","w") as f: f.write("1")
sage: alice.git.silent.add("alices_file")
sage: alice.git.super_silent.commit(message="alice: added alices_file")
```

Alice can now pull the changes by Bob without the need to merge manually:

```
sage: alice.pull()
Merging the remote branch "u/bob/ticket/1" into the local branch "ticket/1".
Automatic merge successful.
```

```
#  (use "sage --dev commit" to commit your merge)
sage: alice.git.echo.log('--pretty=%s')
Merge branch 'u/bob/ticket/1' of ... into ticket/1
alice: added alices_file
bob: added bobs_file
alice: empty commit
initial commit
```

Now, Bob commits some conflicting changes:

```
sage: bob._chdir()
sage: with open("alices_file","w") as f: f.write("2")
sage: bob.git.silent.add("alices_file")
sage: bob.git.super_silent.commit(message="bob: added alices_file")
sage: bob._UI.append('y')
sage: bob.push()
Local commits that are not on the remote branch "u/bob/ticket/1":

    ...: bob: added alices_file

Push to remote branch? [Yes/no] y
```

Now, the pull fails; one would have to use `merge()`:

```
sage: alice._chdir()
sage: alice._UI.append("abort")
sage: alice.pull()
Merging the remote branch "u/bob/ticket/1" into the local branch "ticket/1".
Automatic merge failed, there are conflicting commits.

Auto-merging alices_file
CONFLICT (add/add): Merge conflict in alices_file

Please edit the affected files to resolve the conflicts. When you are finished,
your resolution will be commited.
Finished? [ok/Abort] abort
```

Undo the latest commit by alice, so we can pull again:

```
sage: alice.git.super_silent.reset('HEAD~~', hard=True)
sage: alice.pull()
Merging the remote branch "u/bob/ticket/1" into the local branch "ticket/1".
Automatic merge successful.

#  (use "sage --dev commit" to commit your merge)
sage: alice.git.echo.log('--pretty=%s')
bob: added alices_file
bob: added bobs_file
alice: empty commit
initial commit
```

Now, Alice creates an untracked file which makes a trivial merge impossible:

```
sage: alice._chdir()
sage: open("bobs_other_file","w").close()

sage: bob._chdir()
sage: open("bobs_other_file","w").close()
sage: bob.git.super_silent.add("bobs_other_file")
sage: bob.git.super_silent.commit(message="bob: added bobs_other_file")
```

```
sage: bob._UI.append('y')
sage: bob.push()
Local commits that are not on the remote branch "u/bob/ticket/1":

    ...: bob: added bobs_other_file

Push to remote branch? [Yes/no] y

sage: alice._chdir()
sage: alice._UI.append("abort")
sage: alice.pull()
Merging the remote branch "u/bob/ticket/1" into the local branch "ticket/1".
Automatic merge failed, there are conflicting commits.

Updating ...
error: The following untracked working tree files would be overwritten by merge:
        bobs_other_file
Please move or remove them before you can merge.

Please edit the affected files to resolve the conflicts. When you are finished,
your resolution will be commited.
Finished? [ok/Abort] abort
```

push (*ticket=None*, *remote_branch=None*, *force=False*)

Push the current branch to the Sage repository.

INPUT:

- `ticket` – an integer or string identifying a ticket or `None` (default: `None`), if `None` and currently working on a ticket or if `ticket` specifies a ticket, then the branch on that ticket is set to `remote_branch` after the current branch has been pushed there.
- `remote_branch` – a string or `None` (default: `None`), the remote branch to push to; if `None`, then a default is chosen
- `force` – a boolean (default: `False`), whether to push if this is not a fast-forward.

See Also:

- `commit()` – Save changes to the local repository.
- `pull()` – Update a ticket with changes from the remote repository.

TESTS:

Create a doctest setup with two users:

```
sage: from sage.dev.test.sagedev import two_user_setup
sage: alice, config_alice, bob, config_bob, server = two_user_setup()
```

Alice tries to push to ticket 1 which does not exist yet:

```
sage: alice._chdir()
sage: alice.push(ticket=1)
Ticket name "1" is not valid or ticket does not exist on trac.
```

Alice creates ticket 1 and pushes some changes to it:

```
sage: alice._UI.append("Summary: summary1\ndescription")
sage: alice.create_ticket()
Created ticket #1 at https://trac.sagemath.org/1.
```

```
#  (use "sage --dev checkout --ticket=1" to create a new local branch)
1
sage: alice.checkout(ticket=1)
On ticket #1 with associated local branch "ticket/1".

#  Use "sage --dev merge" to include another ticket/branch.
#  Use "sage --dev commit" to save changes into a new commit.
sage: open("tracked", "w").close()
sage: alice.git.super_silent.add("tracked")
sage: alice.git.super_silent.commit(message="alice: added tracked")
sage: alice._UI.append("y")
sage: alice.push()
The branch "u/alice/ticket/1" does not exist on the remote server.
Create new remote branch? [Yes/no] y
```

Now Bob can check that ticket out and push changes himself:

```
sage: bob._chdir()
sage: bob.checkout(ticket=1)
On ticket #1 with associated local branch "ticket/1".

#  Use "sage --dev merge" to include another ticket/branch.
#  Use "sage --dev commit" to save changes into a new commit.
sage: with open("tracked", "w") as f: f.write("bob")
sage: bob.git.super_silent.add("tracked")
sage: bob.git.super_silent.commit(message="bob: modified tracked")
sage: bob._UI.append("y")
sage: bob._UI.append("y")
sage: bob.push()
The branch "u/bob/ticket/1" does not exist on the remote server.
Create new remote branch? [Yes/no] y
The branch field of ticket #1 needs to be updated from its current value
"u/alice/ticket/1" to "u/bob/ticket/1"
Change the "Branch:" field? [Yes/no] y
```

Now Alice can pull these changes:

```
sage: alice._chdir()
sage: alice.pull()
Merging the remote branch "u/bob/ticket/1" into the local branch "ticket/1".
Automatic merge successful.

#  (use "sage --dev commit" to commit your merge)
```

Alice and Bob make non-conflicting changes simultaneously (and alice adds some characters that need escaping):

```
sage: with open("tracked", "w") as f: f.write("alice")
sage: alice.git.super_silent.add("tracked")
sage: alice.git.super_silent.commit(message="alice: modified tracked {} {{0}} {1}")

sage: bob._chdir()
sage: open("tracked2", "w").close()
sage: bob.git.super_silent.add("tracked2")
sage: bob.git.super_silent.commit(message="bob: added tracked2")
```

After Alice pushed her changes, Bob can not set the branch field anymore:

```
sage: alice._chdir()
sage: alice._UI.append("y")
sage: alice._UI.append("y")
sage: alice.push()
Local commits that are not on the remote branch "u/alice/ticket/1":

    ...: alice: modified tracked {} {{0}} {1}
    ...: bob: modified tracked

Push to remote branch? [Yes/no] y
The branch field of ticket #1 needs to be updated from its current value
"u/bob/ticket/1" to "u/alice/ticket/1"
Change the "Branch:" field? [Yes/no] y

sage: bob._chdir()
sage: bob._UI.append("y")
sage: bob.push()
Local commits that are not on the remote branch "u/bob/ticket/1":

    ....: bob: added tracked2

Push to remote branch? [Yes/no] y
Not setting the branch field for ticket #1 to "u/bob/ticket/1" because
"u/bob/ticket/1" and the current value of the branch field "u/alice/ticket/1"
have diverged.

#  Use "sage --dev pull --ticket=1" to merge the changes introduced by the remote "u/alice/t
```

After merging the changes, this works again:

```
sage: bob.pull()
Merging the remote branch "u/alice/ticket/1" into the local branch "ticket/1".
Automatic merge successful.

#  (use "sage --dev commit" to commit your merge)
sage: bob._UI.append("y")
sage: bob._UI.append("y")
sage: bob.push()
Local commits that are not on the remote branch "u/bob/ticket/1":

    ...: Merge branch 'u/alice/ticket/1' of ... into ticket/1
    ...: alice: modified tracked {} {{0}} {1}

Push to remote branch? [Yes/no] y
The branch field of ticket #1 needs to be updated from its current value
"u/alice/ticket/1" to "u/bob/ticket/1"
Change the "Branch:" field? [Yes/no] y
```

Check that `ticket` works:

```
sage: bob.push(2)
Ticket name "2" is not valid or ticket does not exist on trac.
```

After creating the ticket, this works with a warning:

```
sage: bob._UI.append("Summary: summary2\ndescription")
sage: bob.create_ticket()
Created ticket #2 at https://trac.sagemath.org/2.
```

```
#  (use "sage --dev checkout --ticket=2" to create a new local branch)
2
sage: bob.checkout(ticket=2)
On ticket #2 with associated local branch "ticket/2".

#  Use "sage --dev merge" to include another ticket/branch.
#  Use "sage --dev commit" to save changes into a new commit.
sage: bob.checkout(ticket=1)
On ticket #1 with associated local branch "ticket/1".

#  Use "sage --dev merge" to include another ticket/branch.
#  Use "sage --dev commit" to save changes into a new commit.
sage: bob._UI.append("y")
sage: bob._UI.append("y")
sage: bob.push(2)
About to push the branch "ticket/1" to "u/bob/ticket/2" for ticket #2. However,
your local branch for ticket #2 seems to be "ticket/2".
 Do you really want to proceed? [yes/No] y

#  Use "sage --dev checkout --ticket=2 --branch=ticket/1" to permanently set "ticket/1" as t
The branch "u/bob/ticket/2" does not exist on the remote server.
Create new remote branch? [Yes/no] y
```

Check that `remote_branch` works:

```
sage: bob._UI.append("y")
sage: bob._UI.append("y")
sage: bob.push(remote_branch="u/bob/branch1")
The branch "u/bob/branch1" does not exist on the remote server.
Create new remote branch? [Yes/no] y
The branch field of ticket #1 needs to be updated from its current value
"u/bob/ticket/1" to "u/bob/branch1"
Change the "Branch:" field? [Yes/no] y
```

Check that dependencies are pushed correctly:

```
sage: bob.merge(2)
Merging the remote branch "u/bob/ticket/2" into the local branch "ticket/1".
Automatic merge successful.

#  (use "sage --dev commit" to commit your merge)

Added dependency on #2 to #1.
sage: with open("another_file", "w") as f: f.write("bob after merge(2)")
sage: bob._UI.append('n')
sage: bob.push()
The branch field of ticket #1 needs to be updated from its current value
"u/bob/branch1" to "u/bob/ticket/1"
Change the "Branch:" field? [Yes/no] n
sage: bob._UI.extend(['y', 'y', 'y'])
sage: bob.commit(message="Bob's merge")  # oops
The following files in your working directory are not tracked by git:

    another_file

Start tracking any of these files? [yes/No] y
Start tracking "another_file"? [yes/No] y
Commit your changes to branch "ticket/1"? [Yes/no] y
```

```
#  Use "sage --dev push" to push your commits to the trac server once you are done.
sage: bob._UI.extend(['y', 'y'])
sage: bob.push()
Local commits that are not on the remote branch "u/bob/ticket/1":

    ...: Bob's merge

Push to remote branch? [Yes/no] y
The branch field of ticket #1 needs to be updated from its current value
"u/bob/branch1" to "u/bob/ticket/1"
Change the "Branch:" field? [Yes/no] y
Uploading your dependencies for ticket #1: "" => "#2"
sage: bob._sagedev._set_dependencies_for_ticket(1,())
sage: with open("another_file", "w") as f: f.write("bob after push")
sage: bob._UI.extend(['y', 'y', 'y'])
sage: bob.commit(message='another commit')
Commit your changes to branch "ticket/1"? [Yes/no] y

#  Use "sage --dev push" to push your commits to the trac server once you are done.
sage: bob._UI.extend(['y', "keep", 'y'])
sage: bob.push()
Local commits that are not on the remote branch "u/bob/ticket/1":

    ...: another commit

Push to remote branch? [Yes/no] y
Trac ticket #1 depends on #2 while your local branch depends on no tickets.
Updating dependencies is recommended but optional.
Action for dependencies? [upload/download/keep] keep
sage: with open("another_file", "w") as f: f.write("bob after 2nd push")
sage: bob._UI.append('y')
sage: bob.commit(message='final commit')
Commit your changes to branch "ticket/1"? [Yes/no] y

#  Use "sage --dev push" to push your commits to the trac server once you are done.

sage: bob._UI.extend(['y', 'download', 'y'])
sage: bob.push()
Local commits that are not on the remote branch "u/bob/ticket/1":

    ...: final commit

Push to remote branch? [Yes/no] y
Trac ticket #1 depends on #2 while your local branch depends on no tickets.
Updating dependencies is recommended but optional.
Action for dependencies? [upload/download/keep] download
```

remote_status (*ticket=None*)

Show information about the status of `ticket`.

INPUT:

- `ticket` – an integer or string identifying a ticket or `None` (default: `None`), the number of the ticket to edit. If `None`, show information for the `_current_ticket()`.

TESTS:

Set up a single user for doctesting:

```
sage: from sage.dev.test.sagedev import single_user_setup
sage: dev, config, UI, server = single_user_setup()
```

It is an error to call this without parameters if not on a ticket:

```
sage: dev.remote_status()
ticket must be specified if not currently on a ticket.
```

Create a ticket and show its remote status:

```
sage: UI.append("Summary: ticket1\ndescription")
sage: dev.create_ticket()
Created ticket #1 at https://trac.sagemath.org/1.

#  (use "sage --dev checkout --ticket=1" to create a new local branch)
1
sage: dev.checkout(ticket=1)
On ticket #1 with associated local branch "ticket/1".

#  Use "sage --dev merge" to include another ticket/branch.
#  Use "sage --dev commit" to save changes into a new commit.
sage: dev.remote_status()
Ticket #1 (https://trac.sagemath.org/ticket/1)
==============================================
Your branch "ticket/1" has 0 commits.
No branch has been set on the trac ticket yet.
You have not created a remote branch yet.
```

After pushing the local branch:

```
sage: UI.append("y")
sage: dev.push()
The branch "u/doctest/ticket/1" does not exist on the remote server.
Create new remote branch? [Yes/no] y
sage: dev.remote_status()
Ticket #1 (https://trac.sagemath.org/ticket/1)
==============================================
Your branch "ticket/1" has 0 commits.
The trac ticket points to the branch "u/doctest/ticket/1" which has 0 commits. It does not c
```

Making local changes:

```
sage: open("tracked", "w").close()
sage: dev.git.silent.add("tracked")
sage: dev.git.silent.commit(message="added tracked")
sage: dev.remote_status()
Ticket #1 (https://trac.sagemath.org/ticket/1)
==============================================
Your branch "ticket/1" has 1 commits.
The trac ticket points to the branch "u/doctest/ticket/1" which has 0 commits. "ticket/1" is
...: added tracked
```

Pushing them:

```
sage: UI.append("y")
sage: dev.push()
Local commits that are not on the remote branch "u/doctest/ticket/1":

    ...: added tracked
```

```
Push to remote branch? [Yes/no] y
sage: dev.remote_status()
Ticket #1 (https://trac.sagemath.org/ticket/1)
==============================================
Your branch "ticket/1" has 1 commits.
The trac ticket points to the branch "u/doctest/ticket/1" which has 1 commits. It does not c
```

The branch on the ticket is ahead of the local branch:

```
sage: dev.git.silent.reset('HEAD~', hard=True)
sage: dev.remote_status()
Ticket #1 (https://trac.sagemath.org/ticket/1)
==============================================
Your branch "ticket/1" has 0 commits.
The trac ticket points to the branch "u/doctest/ticket/1" which has 1 commits. "u/doctest/ti
...: added tracked
```

A mixed case:

```
sage: open("tracked2", "w").close()
sage: dev.git.silent.add("tracked2")
sage: dev.git.silent.commit(message="added tracked2")
sage: open("tracked3", "w").close()
sage: dev.git.silent.add("tracked3")
sage: dev.git.silent.commit(message="added tracked3")
sage: open("tracked4", "w").close()
sage: dev.git.silent.add("tracked4")
sage: dev.git.silent.commit(message="added tracked4")
sage: dev._UI.append("y")
sage: dev.push(remote_branch="u/doctest/branch1", force=True)
The branch "u/doctest/branch1" does not exist on the remote server.
Create new remote branch? [Yes/no] y
sage: dev.git.silent.reset('HEAD~', hard=True)
sage: dev.remote_status()
Ticket #1 (https://trac.sagemath.org/ticket/1)
==============================================
Your branch "ticket/1" has 2 commits.
The trac ticket points to the branch "u/doctest/branch1" which has 3 commits. "u/doctest/bra
...: added tracked4
Your remote branch "u/doctest/ticket/1" has 1 commits. The branches "u/doctest/ticket/1" and
"u/doctest/ticket/1" is ahead of "ticket/1" by 1 commits:
...: added tracked
"ticket/1" is ahead of "u/doctest/ticket/1" by 2 commits:
...: added tracked2
...: added tracked3
```

reset_to_clean_state(*error_unless_clean=True*, *helpful=True*)

Reset the current working directory to a clean state.

INPUT:

- `error_unless_clean` – a boolean (default: `True`), whether to raise an `user_interface_error.OperationCancelledError` if the directory remains in an unclean state; used internally.

TESTS:

Set up a single user for doctesting:

```
sage: from sage.dev.test.sagedev import single_user_setup
sage: dev, config, UI, server = single_user_setup()
sage: dev._wrap("reset_to_clean_state")
```

Nothing happens if the directory is already clean:

```
sage: dev.reset_to_clean_state()
```

Bring the directory into a non-clean state:

```
sage: dev.git.super_silent.checkout(b="branch1")
sage: with open("tracked", "w") as f: f.write("boo")
sage: dev.git.silent.add("tracked")
sage: dev.git.silent.commit(message="added tracked")

sage: dev.git.super_silent.checkout('HEAD~')
sage: dev.git.super_silent.checkout(b="branch2")
sage: with open("tracked", "w") as f: f.write("foo")
sage: dev.git.silent.add("tracked")
sage: dev.git.silent.commit(message="added tracked")
sage: from sage.dev.git_error import GitError
sage: try:
....:     dev.git.silent.merge("branch1")
....: except GitError: pass
sage: UI.append("cancel")
sage: dev.reset_to_clean_state()
Repository is in an unclean state (merge). Resetting the state will discard any
uncommited changes.
Reset repository? [reset/Cancel] cancel

#  (use "sage --dev commit" to save changes in a new commit)
sage: UI.append("reset")
sage: dev.reset_to_clean_state()
Repository is in an unclean state (merge). Resetting the state will discard any
uncommited changes.
Reset repository? [reset/Cancel] reset
sage: dev.reset_to_clean_state()
```

A detached HEAD does not count as a non-clean state:

```
sage: dev.git.super_silent.checkout('HEAD', detach=True)
sage: dev.reset_to_clean_state()
```

set_remote(*branch_or_ticket*, *remote_branch*)

Set the remote branch to push to for `branch_or_ticket` to `remote_branch`.

INPUT:

- `branch_or_ticket` – a string, the name of a local branch, or a string or an integer identifying a ticket or `None`; if `None`, the current branch is used.
- `remote_branch` – a string, the name of a remote branch (this branch may not exist yet)

See Also:

- `push()` – To push changes after setting the remote branch

TESTS:

Set up a single user for doctesting:

```
sage: from sage.dev.test.sagedev import single_user_setup
sage: dev, config, UI, server = single_user_setup()
sage: dev._wrap("_remote_branch_for_ticket")
```

Create a new branch:

```
sage: UI.append("Summary: ticket1\ndescription")
sage: dev.create_ticket()
Created ticket #1 at https://trac.sagemath.org/1.

#  (use "sage --dev checkout --ticket=1" to create a new local branch)
1
sage: dev.checkout(ticket=1)
On ticket #1 with associated local branch "ticket/1".

#  Use "sage --dev merge" to include another ticket/branch.
#  Use "sage --dev commit" to save changes into a new commit.
```

Modify the remote branch for this ticket's branch:

```
sage: dev._remote_branch_for_ticket(1)
'u/doctest/ticket/1'
sage: dev.set_remote('ticket/1', 'u/doctest/foo')
sage: dev._remote_branch_for_ticket(1)
'u/doctest/foo'
sage: dev.set_remote('ticket/1', 'foo')
sage: dev._remote_branch_for_ticket(1)
'foo'
sage: dev.set_remote('#1', 'u/doctest/foo')
sage: dev._remote_branch_for_ticket(1)
'u/doctest/foo'
```

show_dependencies(*ticket=None*, *all=False*, *_seen=None*)

Show the dependencies of `ticket`.

INPUT:

- `ticket` – a string or integer identifying a ticket or `None` (default: `None`), the ticket for which dependencies are displayed. If `None`, then the dependencies for the current ticket are displayed.
- `all` – boolean (default: `True`), whether to recursively list all tickets on which this ticket depends (in depth-first order), only including tickets that have a local branch.

Note: Ticket dependencies are stored locally and only updated with respect to the remote server during `push()` and `pull()`.

See Also:

- `TracInterface.dependencies()` – Query Trac to find dependencies.
- `remote_status()` – will show the status of tickets with respect to the remote server.
- `merge()` – Merge in changes from a dependency.
- `diff()` – Show the changes in this branch over the dependencies.

TESTS:

Create a doctest setup with a single user:

```
sage: from sage.dev.test.sagedev import single_user_setup
sage: dev, config, UI, server = single_user_setup()
```

Create some tickets and add dependencies:

```
sage: UI.append("Summary: summary\ndescription")
sage: dev.create_ticket()
Created ticket #1 at https://trac.sagemath.org/1.

#  (use "sage --dev checkout --ticket=1" to create a new local branch)
1
sage: dev.checkout(ticket=1)
On ticket #1 with associated local branch "ticket/1".

#  Use "sage --dev merge" to include another ticket/branch.
#  Use "sage --dev commit" to save changes into a new commit.
sage: UI.append("Summary: summary\ndescription")
sage: dev.create_ticket()
Created ticket #2 at https://trac.sagemath.org/2.

#  (use "sage --dev checkout --ticket=2" to create a new local branch)
2
sage: dev.checkout(ticket=2)
On ticket #2 with associated local branch "ticket/2".

#  Use "sage --dev merge" to include another ticket/branch.
#  Use "sage --dev commit" to save changes into a new commit.
sage: UI.append("Summary: summary\ndescription")
sage: dev.create_ticket()
Created ticket #3 at https://trac.sagemath.org/3.

#  (use "sage --dev checkout --ticket=3" to create a new local branch)
3
sage: dev.checkout(ticket=3)
On ticket #3 with associated local branch "ticket/3".

#  Use "sage --dev merge" to include another ticket/branch.
#  Use "sage --dev commit" to save changes into a new commit.
sage: UI.append("Summary: summary\ndescription")
sage: dev.create_ticket()
Created ticket #4 at https://trac.sagemath.org/4.

#  (use "sage --dev checkout --ticket=4" to create a new local branch)
4
sage: dev.checkout(ticket=4)
On ticket #4 with associated local branch "ticket/4".

#  Use "sage --dev merge" to include another ticket/branch.
#  Use "sage --dev commit" to save changes into a new commit.

sage: dev.merge('ticket/2',create_dependency=True)
Merging the local branch "ticket/2" into the local branch "ticket/4".
Automatic merge successful.

#  (use "sage --dev commit" to commit your merge)

Added dependency on #2 to #4.
sage: dev.merge('ticket/3',create_dependency=True)
```

```
Merging the local branch "ticket/3" into the local branch "ticket/4".
Automatic merge successful.

#  (use "sage --dev commit" to commit your merge)

Added dependency on #3 to #4.
sage: dev.checkout(ticket='#2')
On ticket #2 with associated local branch "ticket/2".

#  Use "sage --dev merge" to include another ticket/branch.
#  Use "sage --dev commit" to save changes into a new commit.
sage: dev.merge('ticket/1', create_dependency=True)
Merging the local branch "ticket/1" into the local branch "ticket/2".
Automatic merge successful.

#  (use "sage --dev commit" to commit your merge)

Added dependency on #1 to #2.
sage: dev.checkout(ticket='#3')
On ticket #3 with associated local branch "ticket/3".

#  Use "sage --dev merge" to include another ticket/branch.
#  Use "sage --dev commit" to save changes into a new commit.
sage: dev.merge('ticket/1', create_dependency=True)
Merging the local branch "ticket/1" into the local branch "ticket/3".
Automatic merge successful.

#  (use "sage --dev commit" to commit your merge)

Added dependency on #1 to #3.
```

Check that the dependencies show correctly:

```
sage: dev.checkout(ticket='#4')
On ticket #4 with associated local branch "ticket/4".

#  Use "sage --dev merge" to include another ticket/branch.
#  Use "sage --dev commit" to save changes into a new commit.
sage: dev.show_dependencies()
Ticket #4 depends on #2, #3.
sage: dev.show_dependencies('#4')
Ticket #4 depends on #2, #3.
sage: dev.show_dependencies('#3')
Ticket #3 depends on #1.
sage: dev.show_dependencies('#2')
Ticket #2 depends on #1.
sage: dev.show_dependencies('#1')
Ticket #1 has no dependencies.
sage: dev.show_dependencies('#4', all=True)
Ticket #4 depends on #3, #1, #2.
```

tickets(*include_abandoned=False*, *cached=True*)

Print the tickets currently being worked on in your local repository.

This function shows the branch names as well as the ticket numbers for all active tickets. It also shows local branches that are not associated to ticket numbers.

INPUT:

- `include_abandoned` – boolean (default: `False`), whether to include abandoned branches.
- `cached` – boolean (default: `True`), whether to try to pull the summaries from the ticket cache; if `True`, then the summaries might not be accurate if they changed since they were last updated. To update the summaries, set this to `False`.

See Also:

- `abandon_ticket()` – hide tickets from this method.
- `remote_status()` – also show status compared to the trac server.
- `current_ticket()` – get the current ticket.

TESTS:

Create a doctest setup with a single user:

```
sage: from sage.dev.test.sagedev import single_user_setup
sage: dev, config, UI, server = single_user_setup()
```

Create some tickets:

```
sage: dev.tickets()
* : master

sage: UI.append("Summary: summary\ndescription")
sage: dev.create_ticket()
Created ticket #1 at https://trac.sagemath.org/1.

#  (use "sage --dev checkout --ticket=1" to create a new local branch)
1
sage: dev.checkout(ticket=1)
On ticket #1 with associated local branch "ticket/1".

#  Use "sage --dev merge" to include another ticket/branch.
#  Use "sage --dev commit" to save changes into a new commit.
sage: UI.append("Summary: summary\ndescription")
sage: dev.create_ticket()
Created ticket #2 at https://trac.sagemath.org/2.

#  (use "sage --dev checkout --ticket=2" to create a new local branch)
2
sage: dev.checkout(ticket=2)
On ticket #2 with associated local branch "ticket/2".

#  Use "sage --dev merge" to include another ticket/branch.
#  Use "sage --dev commit" to save changes into a new commit.
sage: dev.tickets()
    : master
  #1: ticket/1 summary
* #2: ticket/2 summary
```

tmp_dir

A lazy property to provide a temporary directory

TESTS:

```
sage: import os
sage: os.path.isdir(dev._sagedev.tmp_dir)
True
```

upload_ssh_key(*public_key=None*)

Upload `public_key` to gitolite through the trac interface.

INPUT:

- `public_key` – a string or `None` (default: `None`), the path of the key file, defaults to `~/.ssh/id_rsa.pub` (or `~/.ssh/id_dsa.pub` if it exists).

TESTS:

Create a doctest setup with a single user:

```
sage: from sage.dev.test.sagedev import single_user_setup
sage: dev, config, UI, server = single_user_setup()
```

Create and upload a key file:

```
sage: import os
sage: public_key = os.path.join(dev._sagedev.tmp_dir, "id_rsa.pub")
sage: UI.append("no")
sage: UI.append("yes")
sage: dev.upload_ssh_key(public_key=public_key)
The trac git server requires your SSH public key to be able to identify you.
Upload ".../id_rsa.pub" to trac? [Yes/no] yes
File not found: ".../id_rsa.pub"
Create new ssh key pair? [Yes/no] no

#  Use "sage --dev upload-ssh-key" to upload a public key. Or set your key manually at https
sage: UI.append("yes")
sage: UI.append("yes")
sage: dev.upload_ssh_key(public_key=public_key)
The trac git server requires your SSH public key to be able to identify you.
Upload ".../id_rsa.pub" to trac? [Yes/no] yes
File not found: ".../id_rsa.pub"
Create new ssh key pair? [Yes/no] yes
Generating ssh key.
Your key has been uploaded.
sage: UI.append("yes")
sage: dev.upload_ssh_key(public_key=public_key)
The trac git server requires your SSH public key to be able to identify you.
Upload ".../id_rsa.pub" to trac? [Yes/no] yes
Your key has been uploaded.
```

vanilla(*release='master'*)

Return to a clean version of Sage.

INPUT:

- `release` – a string or decimal giving the release name (default: `'master'`). In fact, any tag, commit or branch will work. If the tag does not exist locally an attempt to fetch it from the server will be made.

Git equivalent:

```
Checks out a given tag, commit or branch in detached head mode.
```

See Also:

- `checkout()` – checkout another branch, ready to develop on it.
- `pull()` – pull a branch from the server and merge it.

TESTS:

Create a doctest setup with a single user:

```
sage: from sage.dev.test.sagedev import single_user_setup
sage: dev, config, UI, server = single_user_setup()
```

Go to a sage release:

```
sage: dev.git.current_branch()
'master'
sage: dev.vanilla()
sage: dev.git.current_branch()
Traceback (most recent call last):
...
DetachedHeadError: unexpectedly, git is in a detached HEAD state
```

exception sage.dev.sagedev.**SageDevValueError**(*message*, **args*)

Bases: exceptions.ValueError

A ValueError to indicate that the user supplied an invaid value.

EXAMPLES:

```
sage: from sage.dev.test.sagedev import single_user_setup
sage: dev, config, UI, server = single_user_setup()

sage: dev.checkout(ticket=-1)
Ticket name "-1" is not valid or ticket does not exist on trac.
```

info(**args*)

Store helpful message to be displayed if the exception is not caught.

INPUT:

- **args* – arguments to be passed to info().

OUTPUT:

Returns the exception.

TESTS:

```
sage: from sage.dev.sagedev import SageDevValueError
sage: e = SageDevValueError("message").info('1{0}3', 2)
sage: e.show_info(dev._sagedev._UI)
#  123
```

show_error(*user_interface*)

Display helpful message if available.

INPUT:

- user_interface – an instance of UserInterface.

TESTS:

```
sage: from sage.dev.sagedev import SageDevValueError
sage: e = SageDevValueError("message >{0}<", 123).info('1{0}3', 2)
sage: e.show_error(dev._sagedev._UI)
message >123<
```

show_info(*user_interface*)

Display helpful message if available.

INPUT:

- `user_interface` – an instance of `UserInterface`.

TESTS:

```
sage: from sage.dev.sagedev import SageDevValueError
sage: e = SageDevValueError("message").info('1{0}3', 2)
sage: e.show_info(dev._sagedev._UI)
#  123
```

CHAPTER

TWO

GIT INTERFACE

This module provides a python interface to Sage's git repository.

AUTHORS:

- David Roe, Julian Rueth, Keshav Kini, Nicolas M. Thiery, Robert Bradshaw: initial version

class sage.dev.git_interface.**EchoGitProxy**(*config*, *UI*)
Bases: sage.dev.git_interface.GitProxy

A proxy object to wrap calls to git.

Calls to git show output to stdout and stderr as if the command was executed directly and raise an error on a non-zero exit code.

EXAMPLES:

```
sage: dev.git.echo.status() # not tested
```

class sage.dev.git_interface.**GitInterface**(*config*, *UI*)
Bases: sage.dev.git_interface.ReadStdoutGitProxy

A wrapper around the `git` command line tool.

Most methods of this class correspond to actual git commands. Some add functionality which is not directly available in git. However, all of the methods should be non-interactive. If interaction is required the method should live in `saged.dev.sagedev.SageDev`.

EXAMPLES:

```
sage: from sage.dev.test.config import DoctestConfig
sage: from sage.dev.user_interface import UserInterface
sage: from sage.dev.git_interface import GitInterface
sage: config = DoctestConfig()
sage: GitInterface(config['git'], UserInterface(config['UI']))
GitInterface()
```

clean_wrapper(*remove_untracked_files=False*, *remove_untracked_directories=False*, *remove_ignored=False*)
Clean the working directory.

This is a convenience wrapper for `git clean`

INPUT:

- `remove_untracked_files` – a boolean (default: `False`), whether to remove files which are not tracked by git

- `remove_untracked_directories` – a boolean (default: `False`), whether to remove directories which are not tracked by git
- `remove_ignored` – a boolean (default: `False`), whether to remove files directories which are ignored by git

EXAMPLES:

Create a `GitInterface` for doctesting:

```
sage: from sage.dev.git_interface import GitInterface
sage: from sage.dev.test.config import DoctestConfig
sage: from sage.dev.test.user_interface import DoctestUserInterface
sage: config = DoctestConfig()
sage: git = GitInterface(config["git"], DoctestUserInterface(config["UI"]))
```

Set up some files/directories:

```
sage: os.chdir(config['git']['src'])
sage: open('tracked','w').close()
sage: git.silent.add('tracked')
sage: with open('.gitignore','w') as f: f.write('ignored\nignored_dir')
sage: git.silent.add('.gitignore')
sage: git.silent.commit('-m', 'initial commit')

sage: os.mkdir('untracked_dir')
sage: open('untracked_dir/untracked','w').close()
sage: open('untracked','w').close()
sage: open('ignored','w').close()
sage: os.mkdir('ignored_dir')
sage: open('ignored_dir/untracked','w').close()
sage: with open('tracked','w') as f: f.write('version 0')
sage: git.echo.status()
# On branch master
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
#   modified:   tracked
#
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#   untracked
#   untracked_dir/
no changes added to commit (use "git add" and/or "git commit -a")
```

Some invalid combinations of flags:

```
sage: git.clean_wrapper(
....:     remove_untracked_files=False, remove_untracked_directories=True)
Traceback (most recent call last):
...
ValueError: remove_untracked_directories only valid if remove_untracked_files is set
sage: git.clean_wrapper(remove_untracked_files = False, remove_ignored = True)
Traceback (most recent call last):
...
ValueError: remove_ignored only valid if remove_untracked_files is set
```

Per default only the tracked modified files are reset to a clean state:

```
sage: git.clean_wrapper()
sage: git.echo.status()
# On branch master
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#   untracked
#   untracked_dir/
nothing added to commit but untracked files present (use "git add" to track)
```

Untracked items can be removed by setting the parameters:

```
sage: git.clean_wrapper(remove_untracked_files=True)
Removing untracked
sage: git.clean_wrapper(
....:     remove_untracked_files=True, remove_untracked_directories=True)
Removing untracked_dir/
sage: git.clean_wrapper(
....:     remove_untracked_files=True, remove_ignored=True)
Removing ignored
sage: git.clean_wrapper(
....:     remove_untracked_files=True,
....:     remove_untracked_directories=True,
....:     remove_ignored=True)
Removing ignored_dir/
```

commit_for_branch(*branch*)

Return the commit id of the local `branch`, or `None` if the branch does not exist

EXAMPLES:

Create a `GitInterface` for doctesting:

```
sage: import os
sage: from sage.dev.git_interface import GitInterface
sage: from sage.dev.test.config import DoctestConfig
sage: from sage.dev.test.user_interface import DoctestUserInterface
sage: config = DoctestConfig()
sage: git = GitInterface(config["git"], DoctestUserInterface(config["UI"]))
```

Create some branches:

```
sage: os.chdir(config['git']['src'])
sage: git.silent.commit('-m','initial commit','--allow-empty')
sage: git.silent.branch('branch1')
sage: git.silent.branch('branch2')
```

Check existence of branches:

```
sage: git.commit_for_branch('branch1') # random output
'087e1fdd0fe6f4c596f5db22bc54567b032f5d2b'
sage: git.commit_for_branch('branch2') is not None
True
sage: git.commit_for_branch('branch3') is not None
False
```

commit_for_ref(*ref*)

Return the commit id of the `ref`, or `None` if the `ref` does not exist.

EXAMPLES:

Create a `GitInterface` for doctesting:

```
sage: import os
sage: from sage.dev.git_interface import GitInterface
sage: from sage.dev.test.config import DoctestConfig
sage: from sage.dev.test.user_interface import DoctestUserInterface
sage: config = DoctestConfig()
sage: git = GitInterface(config["git"], DoctestUserInterface(config["UI"]))
```

Create some branches:

```
sage: os.chdir(config['git']['src'])
sage: git.silent.commit('-m','initial commit','--allow-empty')
sage: git.silent.branch('branch1')
sage: git.silent.branch('branch2')
```

Check existence of branches:

```
sage: git.commit_for_ref('refs/heads/branch1') # random output
'087e1fdd0fe6f4c596f5db22bc54567b032f5d2b'
sage: git.commit_for_ref('refs/heads/branch2') is not None
True
sage: git.commit_for_ref('refs/heads/branch3') is not None
False
```

current_branch()

Return the current branch

EXAMPLES:

Create a `GitInterface` for doctesting:

```
sage: import os
sage: from sage.dev.git_interface import GitInterface
sage: from sage.dev.test.config import DoctestConfig
sage: from sage.dev.test.user_interface import DoctestUserInterface
sage: config = DoctestConfig()
sage: git = GitInterface(config["git"], DoctestUserInterface(config["UI"]))
```

Create some branches:

```
sage: os.chdir(config['git']['src'])
sage: git.silent.commit('-m','initial commit','--allow-empty')
sage: git.silent.commit('-m','second commit','--allow-empty')
sage: git.silent.branch('branch1')
sage: git.silent.branch('branch2')

sage: git.current_branch()
'master'
sage: git.super_silent.checkout('branch1')
sage: git.current_branch()
'branch1'
```

If `HEAD` is detached:

```
sage: git.super_silent.checkout('master~')
sage: git.current_branch()
Traceback (most recent call last):
...
DetachedHeadError: unexpectedly, git is in a detached HEAD state
```

get_state()

Get the current state of merge/rebase/am/etc operations.

OUTPUT:

A tuple of strings which consists of any of the following: 'rebase', 'am', 'rebase-i', 'rebase-m','merge','bisect','cherry-seq','cherry'.

EXAMPLES:

Create a GitInterface for doctesting:

```
sage: import os
sage: from sage.dev.git_interface import GitInterface
sage: from sage.dev.test.config import DoctestConfig
sage: from sage.dev.test.user_interface import DoctestUserInterface
sage: config = DoctestConfig()
sage: git = GitInterface(config["git"], DoctestUserInterface(config["UI"]))
```

Create two conflicting branches:

```
sage: os.chdir(config['git']['src'])
sage: with open("file","w") as f: f.write("version 0")
sage: git.silent.add("file")
sage: git.silent.commit("-m","initial commit")
sage: git.super_silent.checkout("-b","branch1")
sage: with open("file","w") as f: f.write("version 1")
sage: git.silent.commit("-am","second commit")
sage: git.super_silent.checkout("master")
sage: git.super_silent.checkout("-b","branch2")
sage: with open("file","w") as f: f.write("version 2")
sage: git.silent.commit("-am","conflicting commit")
```

A merge state:

```
sage: git.super_silent.checkout("branch1")
sage: git.silent.merge('branch2')
Traceback (most recent call last):
...
GitError: git returned with non-zero exit code (1) for
"git -c user.email=doc@test.test -c user.name=doctest merge branch2".
...
sage: git.get_state()
('merge',)
sage: git.silent.merge(abort=True)
sage: git.get_state()
()
```

A rebase state:

```
sage: git._execute_supersilent('rebase', 'branch2')
Traceback (most recent call last):
...
GitError: git returned with non-zero exit code (1) for
"git -c user.email=doc@test.test -c user.name=doctest rebase branch2".
...
sage: git.get_state()
('rebase',)
sage: git.super_silent.rebase(abort=True)
sage: git.get_state()
()
```

has_uncommitted_changes()

Return whether there are uncommitted changes, i.e., whether there are modified files which are tracked by git.

EXAMPLES:

Create a `GitInterface` for doctesting:

```
sage: import os
sage: from sage.dev.git_interface import GitInterface
sage: from sage.dev.test.config import DoctestConfig
sage: from sage.dev.test.user_interface import DoctestUserInterface
sage: config = DoctestConfig()
sage: git = GitInterface(config["git"], DoctestUserInterface(config["UI"]))
```

An untracked file does not count towards uncommited changes:

```
sage: os.chdir(config['git']['src'])
sage: open('untracked','w').close()
sage: git.has_uncommitted_changes()
False
```

Once added to the index it does:

```
sage: git.silent.add('untracked')
sage: git.has_uncommitted_changes()
True
sage: git.silent.commit('-m', 'tracking untracked')
sage: git.has_uncommitted_changes()
False
sage: with open('untracked','w') as f: f.write('version 0')
sage: git.has_uncommitted_changes()
True
```

is_ancestor_of(*a*, *b*)

Return whether `a` is an ancestor of `b`.

EXAMPLES:

Create a `GitInterface` for doctesting:

```
sage: import os
sage: from sage.dev.git_interface import GitInterface
sage: from sage.dev.test.config import DoctestConfig
sage: from sage.dev.test.user_interface import DoctestUserInterface
sage: config = DoctestConfig()
sage: git = GitInterface(config["git"], DoctestUserInterface(config["UI"]))
```

Create two conflicting branches:

```
sage: os.chdir(config['git']['src'])
sage: with open("file","w") as f: f.write("version 0")
sage: git.silent.add("file")
sage: git.silent.commit("-m","initial commit")
sage: git.super_silent.checkout("-b","branch1")
sage: with open("file","w") as f: f.write("version 1")
sage: git.silent.commit("-am","second commit")
sage: git.super_silent.checkout("master")
sage: git.super_silent.checkout("-b","branch2")
sage: with open("file","w") as f: f.write("version 2")
sage: git.silent.commit("-am","conflicting commit")

sage: git.is_ancestor_of('master', 'branch2')
```

```
True
sage: git.is_ancestor_of('branch2', 'master')
False
sage: git.is_ancestor_of('branch1', 'branch2')
False
sage: git.is_ancestor_of('master', 'master')
True
```

is_child_of(*a*, *b*)

Return whether `a` is a child of `b`.

EXAMPLES:

Create a `GitInterface` for doctesting:

```
sage: import os
sage: from sage.dev.git_interface import GitInterface
sage: from sage.dev.test.config import DoctestConfig
sage: from sage.dev.test.user_interface import DoctestUserInterface
sage: config = DoctestConfig()
sage: git = GitInterface(config["git"], DoctestUserInterface(config["UI"]))
```

Create two conflicting branches:

```
sage: os.chdir(config['git']['src'])
sage: with open("file","w") as f: f.write("version 0")
sage: git.silent.add("file")
sage: git.silent.commit("-m","initial commit")
sage: git.super_silent.checkout("-b","branch1")
sage: with open("file","w") as f: f.write("version 1")
sage: git.silent.commit("-am","second commit")
sage: git.super_silent.checkout("master")
sage: git.super_silent.checkout("-b","branch2")
sage: with open("file","w") as f: f.write("version 2")
sage: git.silent.commit("-am","conflicting commit")

sage: git.is_child_of('master', 'branch2')
False
sage: git.is_child_of('branch2', 'master')
True
sage: git.is_child_of('branch1', 'branch2')
False
sage: git.is_child_of('master', 'master')
True
```

local_branches()

Return a list of local branches sorted by last commit time.

EXAMPLES:

Create a `GitInterface` for doctesting:

```
sage: import os, time
sage: from sage.dev.git_interface import GitInterface
sage: from sage.dev.test.config import DoctestConfig
sage: from sage.dev.test.user_interface import DoctestUserInterface
sage: config = DoctestConfig()
sage: git = GitInterface(config["git"], DoctestUserInterface(config["UI"]))
```

Create some branches:

```
sage: os.chdir(config['git']['src'])
sage: env = {'GIT_COMMITTER_DATE': time.strftime("%Y-%m-%dT%H:%M:10")}
sage: git.silent.commit('-m','initial commit','--allow-empty', env=env)
sage: git.super_silent.checkout('-b', 'branch')
sage: env['GIT_COMMITTER_DATE'] = time.strftime("%Y-%m-%dT%H:%M:20")
sage: git.silent.commit('-m','second commit','--allow-empty', env=env)
sage: git.super_silent.checkout('-b', 'other', 'master')
sage: env['GIT_COMMITTER_DATE'] = time.strftime("%Y-%m-%dT%H:%M:30")
sage: git.silent.commit('-m','third commit','--allow-empty', env=env)
```

Use this repository as a remote repository:

```
sage: config2 = DoctestConfig()
sage: git2 = GitInterface(config2["git"], DoctestUserInterface(config["UI"]))
sage: os.chdir(config2['git']['src'])
sage: env['GIT_COMMITTER_DATE'] = time.strftime("%Y-%m-%dT%H:%M:40")
sage: git2.silent.commit('-m','initial commit','--allow-empty', env=env)
sage: git2.silent.remote('add', 'git', config['git']['src'])
sage: git2.super_silent.fetch('git')
sage: git2.super_silent.checkout("branch")
sage: git2.echo.branch("-a")
* branch
  master
  remotes/git/branch
  remotes/git/master
  remotes/git/other

sage: git2.local_branches()
['master', 'branch']
sage: os.chdir(config['git']['src'])
sage: git.local_branches()
['other', 'branch', 'master']
```

rename_branch(*oldname*, *newname*)

Rename `oldname` to `newname`.

EXAMPLES:

Create a `GitInterface` for doctesting:

```
sage: from sage.dev.git_interface import GitInterface
sage: from sage.dev.test.config import DoctestConfig
sage: from sage.dev.test.user_interface import DoctestUserInterface
sage: config = DoctestConfig()
sage: git = GitInterface(config["git"], DoctestUserInterface(config["UI"]))
```

Create some branches:

```
sage: os.chdir(config['git']['src'])
sage: git.silent.commit('-m','initial commit','--allow-empty')
sage: git.silent.branch('branch1')
sage: git.silent.branch('branch2')
```

Rename some branches:

```
sage: git.rename_branch('branch1', 'branch3')
sage: git.rename_branch('branch2', 'branch3')
Traceback (most recent call last):
...
GitError: git returned with non-zero exit code (128) for
```

```
"git -c user.email=doc@test.test -c user.name=doctest branch --move branch2 branch3".
output to stderr: fatal: A branch named 'branch3' already exists.
```

reset_to_clean_state()

Get out of a merge/am/rebase/etc state.

EXAMPLES:

Create a GitInterface for doctesting:

```
sage: import os
sage: from sage.dev.git_interface import GitInterface
sage: from sage.dev.test.config import DoctestConfig
sage: from sage.dev.test.user_interface import DoctestUserInterface
sage: config = DoctestConfig()
sage: git = GitInterface(config["git"], DoctestUserInterface(config["UI"]))
```

Create two conflicting branches:

```
sage: os.chdir(config['git']['src'])
sage: with open("file","w") as f: f.write("version 0")
sage: git.silent.add("file")
sage: git.silent.commit("-m","initial commit")
sage: git.super_silent.checkout("-b","branch1")
sage: with open("file","w") as f: f.write("version 1")
sage: git.silent.commit("-am","second commit")
sage: git.super_silent.checkout("master")
sage: git.super_silent.checkout("-b","branch2")
sage: with open("file","w") as f: f.write("version 2")
sage: git.silent.commit("-am","conflicting commit")
```

A merge:

```
sage: git.silent.merge('branch1')
Traceback (most recent call last):
...
GitError: git returned with non-zero exit code (1) for
"git -c user.email=doc@test.test -c user.name=doctest merge branch1".
...
sage: git.get_state()
('merge',)
```

Get out of this state:

```
sage: git.reset_to_clean_state()
sage: git.get_state()
()
```

untracked_files()

Return a list of file names for files that are not tracked by git and not ignored.

EXAMPLES:

Create a GitInterface for doctesting:

```
sage: import os
sage: from sage.dev.git_interface import GitInterface
sage: from sage.dev.test.config import DoctestConfig
sage: from sage.dev.test.user_interface import DoctestUserInterface
sage: config = DoctestConfig()
sage: git = GitInterface(config["git"], DoctestUserInterface(config["UI"]))
```

An untracked file:

```
sage: os.chdir(config['git']['src'])
sage: git.untracked_files()
[]
sage: open('untracked','w').close()
sage: git.untracked_files()
['untracked']
```

Directories are not displayed here::

```
sage: os.mkdir('untracked_dir')
sage: git.untracked_files()
['untracked']
sage: open('untracked_dir/untracked','w').close()
sage: git.untracked_files()
['untracked', 'untracked_dir/untracked']
```

class sage.dev.git_interface.**GitProxy**(*config*, *UI*)

Bases: object

A proxy object to wrap actual calls to git.

EXAMPLES:

```
sage: from sage.dev.git_interface import GitProxy
sage: from sage.dev.test.config import DoctestConfig
sage: from sage.dev.test.user_interface import DoctestUserInterface
sage: config = DoctestConfig()
sage: GitProxy(config['git'], DoctestUserInterface(config['UI']))
<sage.dev.git_interface.GitProxy object at 0x...>
```

add(**args*, ***kwds*)

Call git add.

OUTPUT:

See the docstring of __call__ for more information.

EXAMPLES:

sage: dev.git.add() # not tested

am(**args*, ***kwds*)

Call git am.

OUTPUT:

See the docstring of __call__ for more information.

EXAMPLES:

sage: dev.git.am() # not tested

apply(**args*, ***kwds*)

Call git apply.

OUTPUT:

See the docstring of __call__ for more information.

EXAMPLES:

sage: dev.git.apply() # not tested

bisect(**args*, ***kwds*)
Call `git bisect`.

OUTPUT:

See the docstring of `__call__` for more information.

EXAMPLES:

sage: dev.git.bisect() # not tested

branch(**args*, ***kwds*)
Call `git branch`.

OUTPUT:

See the docstring of `__call__` for more information.

EXAMPLES:

sage: dev.git.branch() # not tested

checkout(**args*, ***kwds*)
Call `git checkout`.

OUTPUT:

See the docstring of `__call__` for more information.

EXAMPLES:

sage: dev.git.checkout() # not tested

cherry_pick(**args*, ***kwds*)
Call `git cherry-pick`.

OUTPUT:

See the docstring of `__call__` for more information.

EXAMPLES:

sage: dev.git.cherry_pick() # not tested

clean(**args*, ***kwds*)
Call `git clean`.

OUTPUT:

See the docstring of `__call__` for more information.

EXAMPLES:

sage: dev.git.clean() # not tested

clone(**args*, ***kwds*)
Call `git clone`.

OUTPUT:

See the docstring of `__call__` for more information.

EXAMPLES:

sage: dev.git.clone() # not tested

commit(**args*, ***kwds*)
Call `git commit`.

OUTPUT:

See the docstring of `__call__` for more information.

EXAMPLES:

sage: dev.git.commit() # not tested

config(**args*, ***kwds*)
Call `git config`.

OUTPUT:

See the docstring of `__call__` for more information.

EXAMPLES:

sage: dev.git.config() # not tested

diff(**args*, ***kwds*)
Call `git diff`.

OUTPUT:

See the docstring of `__call__` for more information.

EXAMPLES:

sage: dev.git.diff() # not tested

fetch(**args*, ***kwds*)
Call `git fetch`.

OUTPUT:

See the docstring of `__call__` for more information.

EXAMPLES:

sage: dev.git.fetch() # not tested

for_each_ref(**args*, ***kwds*)
Call `git for-each-ref`.

OUTPUT:

See the docstring of `__call__` for more information.

EXAMPLES:

sage: dev.git.for_each_ref() # not tested

format_patch(**args*, ***kwds*)
Call `git format-patch`.

OUTPUT:

See the docstring of `__call__` for more information.

EXAMPLES:

sage: dev.git.format_patch() # not tested

grep(**args*, ***kwds*)
Call `git grep`.

OUTPUT:

See the docstring of `__call__` for more information.

EXAMPLES:

sage: dev.git.grep() # not tested

init(**args*, ***kwds*)
Call `git init`.

OUTPUT:

See the docstring of `__call__` for more information.

EXAMPLES:

sage: dev.git.init() # not tested

log(**args*, ***kwds*)
Call `git log`.

OUTPUT:

See the docstring of `__call__` for more information.

EXAMPLES:

sage: dev.git.log() # not tested

ls_files(**args*, ***kwds*)
Call `git ls-files`.

OUTPUT:

See the docstring of `__call__` for more information.

EXAMPLES:

sage: dev.git.ls_files() # not tested

ls_remote(**args*, ***kwds*)
Call `git ls-remote`.

OUTPUT:

See the docstring of `__call__` for more information.

EXAMPLES:

sage: dev.git.ls_remote() # not tested

merge(**args*, ***kwds*)
Call `git merge`.

OUTPUT:

See the docstring of `__call__` for more information.

EXAMPLES:

sage: dev.git.merge() # not tested

merge_base(**args*, ***kwds*)
Call `git merge-base`.

OUTPUT:

See the docstring of `__call__` for more information.

EXAMPLES:

sage: dev.git.merge_base() # not tested

mv(**args*, ***kwds*)
Call `git mv`.

OUTPUT:

See the docstring of `__call__` for more information.

EXAMPLES:

sage: dev.git.mv() # not tested

pull(**args*, ***kwds*)
Call `git pull`.

OUTPUT:

See the docstring of `__call__` for more information.

EXAMPLES:

sage: dev.git.pull() # not tested

push(**args*, ***kwds*)
Call `git push`.

OUTPUT:

See the docstring of `__call__` for more information.

EXAMPLES:

sage: dev.git.push() # not tested

rebase(**args*, ***kwds*)
Call `git rebase`.

OUTPUT:

See the docstring of `__call__` for more information.

EXAMPLES:

sage: dev.git.rebase() # not tested

remote(**args*, ***kwds*)
Call `git remote`.

OUTPUT:

See the docstring of `__call__` for more information.

EXAMPLES:

sage: dev.git.remote() # not tested

reset(**args*, ***kwds*)
Call `git reset`.

OUTPUT:

See the docstring of `__call__` for more information.

EXAMPLES:

sage: dev.git.reset() # not tested

rev_list(**args*, ***kwds*)
Call `git rev-list`.

OUTPUT:

See the docstring of `__call__` for more information.

EXAMPLES:

sage: dev.git.rev_list() # not tested

rev_parse(**args*, ***kwds*)
Call `git rev-parse`.

OUTPUT:

See the docstring of `__call__` for more information.

EXAMPLES:

sage: dev.git.rev_parse() # not tested

rm(**args*, ***kwds*)
Call `git rm`.

OUTPUT:

See the docstring of `__call__` for more information.

EXAMPLES:

sage: dev.git.rm() # not tested

show(**args*, ***kwds*)
Call `git show`.

OUTPUT:

See the docstring of `__call__` for more information.

EXAMPLES:

sage: dev.git.show() # not tested

show_ref(**args*, ***kwds*)
Call `git show-ref`.

OUTPUT:

See the docstring of `__call__` for more information.

EXAMPLES:

sage: dev.git.show_ref() # not tested

stash (**args*, ***kwds*)
Call `git stash`.

OUTPUT:

See the docstring of `__call__` for more information.

EXAMPLES:

sage: dev.git.stash() # not tested

status (**args*, ***kwds*)
Call `git status`.

OUTPUT:

See the docstring of `__call__` for more information.

EXAMPLES:

sage: dev.git.status() # not tested

symbolic_ref (**args*, ***kwds*)
Call `git symbolic-ref`.

OUTPUT:

See the docstring of `__call__` for more information.

EXAMPLES:

sage: dev.git.symbolic_ref() # not tested

tag (**args*, ***kwds*)
Call `git tag`.

OUTPUT:

See the docstring of `__call__` for more information.

EXAMPLES:

sage: dev.git.tag() # not tested

class `sage.dev.git_interface.`**ReadStdoutGitProxy**(*config*, *UI*)
Bases: `sage.dev.git_interface.GitProxy`

A proxy object to wrap calls to git.

Calls to git return the stdout of git and raise an error on a non-zero exit code. Output to stderr is supressed.

EXAMPLES:

```
sage: dev.git.status() # not tested
```

class `sage.dev.git_interface.`**SilentGitProxy**(*config*, *UI*)
Bases: `sage.dev.git_interface.GitProxy`

A proxy object to wrap calls to git.

Calls to git do not show any output to stdout and raise an error on a non-zero exit code. Output to stderr is printed.

EXAMPLES:

```
sage: dev.git.silent.status() # not tested
```

class sage.dev.git_interface.**SuperSilentGitProxy**(*config*, *UI*)
Bases: sage.dev.git_interface.GitProxy

A proxy object to wrap calls to git.

Calls to git do not show any output to stderr or stdout and raise an error on a non-zero exit code.

EXAMPLES:

```
sage: dev.git.super_silent.status() # not tested
```

sage.dev.git_interface.**create_wrapper**(*git_cmd__*)
Create a wrapper for git_cmd__.

EXAMPLES:

```
sage: import os
sage: from sage.dev.git_interface import GitInterface
sage: from sage.dev.test.config import DoctestConfig
sage: from sage.dev.test.user_interface import DoctestUserInterface
sage: config = DoctestConfig()
sage: git = GitInterface(config["git"], DoctestUserInterface(config["UI"]))
sage: os.chdir(config['git']['src'])
sage: git.echo.status() # indirect doctest
# On branch master
#
# Initial commit
#
nothing to commit (create/copy files and use "git add" to track)
```

CHAPTER

THREE

TRAC INTERFACE

This module provides an interface to access sage's issue tracker 'trac' through its RPC interface.

AUTHORS:

- David Roe, Julian Rueth, R. Andrew Ohana, Robert Bradshaw, Timo Kluck: initial version

exception sage.dev.trac_interface.**TicketSyntaxError**

Bases: exceptions.SyntaxError

A syntax error when parsing a ticket description modified by the user.

EXAMPLES:

```
sage: from sage.dev.trac_interface import TicketSyntaxError
sage: raise TicketSyntaxError()
Traceback (most recent call last):
...
TicketSyntaxError: None
```

class sage.dev.trac_interface.**TracInterface**(*config*, *UI*)

Bases: object

Wrapper around the XML-RPC interface of trac.

EXAMPLES:

```
sage: from sage.dev.test.config import DoctestConfig
sage: from sage.dev.test.user_interface import DoctestUserInterface
sage: from sage.dev.trac_interface import TracInterface
sage: config = DoctestConfig()
sage: trac = TracInterface(config['trac'], DoctestUserInterface(config['UI']))
sage: trac
<sage.dev.trac_interface.TracInterface object at 0x...>
```

add_comment(*ticket*, *comment*)

Add comment to ticket on trac.

See Also:

add_comment_interactive()

EXAMPLES:

```
sage: from sage.dev.test.config import DoctestConfig
sage: from sage.dev.test.user_interface import DoctestUserInterface
sage: from sage.dev.test.trac_interface import DoctestTracInterface
sage: from sage.dev.test.trac_server import DoctestTracServer
sage: config = DoctestConfig()
```

```
sage: config['trac']['password'] = 'secret'
sage: UI = DoctestUserInterface(config['UI'])
sage: trac = DoctestTracInterface(config['trac'], UI, DoctestTracServer())
sage: ticket = trac.create_ticket('Summary', 'Description', {'type':'defect', 'component':'a
sage: trac.add_comment(ticket, "a comment")
```

add_comment_interactive (*ticket*, *comment=''*)

Add a comment to `ticket` on trac.

INPUT:

- `comment` – a string (default: ''), the default value for the comment to add.

EXAMPLES:

```
sage: from sage.dev.test.config import DoctestConfig
sage: from sage.dev.test.user_interface import DoctestUserInterface
sage: from sage.dev.test.trac_interface import DoctestTracInterface
sage: from sage.dev.test.trac_server import DoctestTracServer
sage: config = DoctestConfig()
sage: config['trac']['password'] = 'secret'
sage: UI = DoctestUserInterface(config['UI'])
sage: trac = DoctestTracInterface(config['trac'], UI, DoctestTracServer())
sage: ticket = trac.create_ticket('Summary', 'Description', {'type':'defect', 'component':'a

sage: UI.append("# empty comment")
sage: trac.add_comment_interactive(ticket)
Traceback (most recent call last):
...
OperationCancelledError: comment creation aborted

sage: UI.append("a comment")
sage: trac.add_comment_interactive(ticket)
```

attachment_names (*ticket*)

Retrieve the names of the attachments for `ticket`.

EXAMPLES:

```
sage: dev.trac.attachment_names(1000) # optional: internet
()
sage: dev.trac.attachment_names(13147) # optional: internet
('13147_move.patch',
 '13147_lazy.patch',
 '13147_lazy_spkg.patch',
 '13147_new.patch',
 '13147_over_13579.patch',
 'trac_13147-ref.patch',
 'trac_13147-rebased-to-13681.patch',
 'trac_13681_root.patch')
```

create_ticket (*summary*, *description*, *attributes={}*)

Create a ticket on trac and return the new ticket number.

See Also:

`create_ticket_interactive()`

EXAMPLES:

```
sage: from sage.dev.test.config import DoctestConfig
sage: from sage.dev.test.user_interface import DoctestUserInterface
sage: from sage.dev.test.trac_interface import DoctestTracInterface
sage: from sage.dev.test.trac_server import DoctestTracServer
sage: config = DoctestConfig()
sage: config['trac']['password'] = 'secret'
sage: UI = DoctestUserInterface(config['UI'])
sage: trac = DoctestTracInterface(config['trac'], UI, DoctestTracServer())
sage: trac.create_ticket('Summary', 'Description', {'type':'defect', 'component':'algebra'})
1
```

create_ticket_interactive()

Drop user into an editor for creating a ticket.

EXAMPLE:

```
sage: from sage.dev.test.config import DoctestConfig
sage: from sage.dev.test.user_interface import DoctestUserInterface
sage: from sage.dev.test.trac_interface import DoctestTracInterface
sage: from sage.dev.test.trac_server import DoctestTracServer
sage: config = DoctestConfig()
sage: config['trac']['password'] = 'secret'
sage: UI = DoctestUserInterface(config['UI'])
sage: trac = DoctestTracInterface(config['trac'], UI, DoctestTracServer())
sage: UI.append("Summary: summary\nType: defect\nPriority: minor\nComponent: algebra\ndescri
sage: trac.create_ticket_interactive()
1
```

dependencies(*ticket*, *recurse=False*, *seen=None*)

Retrieve dependencies of `ticket`, sorted by ticket number.

INPUT:

- `ticket` – an integer, the number of the ticket
- `recurse` – a boolean (default: `False`), whether to get indirect dependencies of `ticket`
- `seen` – a list (default: `[]`), used internally to implement `recurse`

EXAMPLES:

```
sage: from sage.dev.test.sagedev import single_user_setup_with_internet
sage: dev = single_user_setup_with_internet()[0]
sage: dev.trac.dependencies(1000)            # optional: internet (an old ticket with no dep
[]
sage: dev.trac.dependencies(13147)           # optional: internet
[13579, 13681]
sage: dev.trac.dependencies(13147, recurse=True) # long time, optional: internet
[13579, 13631, 13681]
```

edit_ticket_interactive(*ticket*)

Edit `ticket` on trac.

EXAMPLES:

```
sage: from sage.dev.test.config import DoctestConfig
sage: from sage.dev.test.user_interface import DoctestUserInterface
sage: from sage.dev.test.trac_interface import DoctestTracInterface
sage: from sage.dev.test.trac_server import DoctestTracServer
sage: config = DoctestConfig()
sage: config['trac']['password'] = 'secret'
```

```
sage: UI = DoctestUserInterface(config['UI'])
sage: trac = DoctestTracInterface(config['trac'], UI, DoctestTracServer())
sage: ticket = trac.create_ticket('Summary', 'Description', {'type':'defect', 'component':'a

sage: UI.append("# empty")
sage: trac.edit_ticket_interactive(ticket)
Traceback (most recent call last):
...
OperationCancelledError: ticket edit aborted

sage: UI.append("Summary: summary\ndescription\n")
sage: trac.edit_ticket_interactive(ticket)
```

query(*qstr*)

Return a list of ticket ids that match the given query string.

INPUT:

- `qstr` – a query string. All queries will use stored settings for maximum number of results per page and paging options. Use max=n to define number of results to receive, and use page=n to page through larger result sets. Using max=0 will turn off paging and return all results.

EXAMPLES:

```
sage: dev.trac.query('status!=closed&(component=padics||component=misc)&max=3') # random, op
[329, 15130, 21]
```

reset_password()

Reset password stored in this object and in the configuration.

EXAMPLES:

```
sage: from sage.dev.test.config import DoctestConfig
sage: from sage.dev.test.user_interface import DoctestUserInterface
sage: from sage.dev.trac_interface import TracInterface
sage: config = DoctestConfig()
sage: UI = DoctestUserInterface(config['UI'])
sage: trac = TracInterface(config['trac'], UI)
sage: UI.append('y')
sage: UI.append('secret')
sage: trac._password
Trac password:
You can save your password in a configuration file. However, this file might be
readable by privileged users on this system.
Save password in file? [yes/No] y
#  Your trac password has been written to a configuration file. To reset your
password, use "dev.trac.reset_password()".
'secret'
sage: config['trac']['password']
'secret'
sage: trac.reset_password()
sage: config['trac']['password']
Traceback (most recent call last):
...
KeyError: 'password'
```

reset_username()

Reset username and password stored in this object and in the configuration.

EXAMPLES:

```
sage: from sage.dev.test.config import DoctestConfig
sage: from sage.dev.test.user_interface import DoctestUserInterface
sage: from sage.dev.trac_interface import TracInterface
sage: config = DoctestConfig()
sage: UI = DoctestUserInterface(config['UI'])
sage: trac = TracInterface(config['trac'], UI)
sage: trac.reset_username()
sage: UI.append("doctest2")
sage: trac._username
Trac username: doctest2
#  Your trac username has been written to a configuration file for future
sessions. To reset your username, use "dev.trac.reset_username()".
'doctest2'
```

set_attributes(*ticket*, *comment=''*, *notify=False*, ***kwds*)

Set attributes on a track ticket.

INPUT:

- `ticket` – the ticket id
- `comment` – a comment when changing these attributes
- `kwds` – a dictionary of field:value pairs

See Also:

`_get_attributes()`

EXAMPLES:

```
sage: from sage.dev.test.config import DoctestConfig
sage: from sage.dev.test.user_interface import DoctestUserInterface
sage: from sage.dev.test.trac_interface import DoctestTracInterface
sage: from sage.dev.test.trac_server import DoctestTracServer
sage: config = DoctestConfig()
sage: config['trac']['password'] = 'secret'
sage: UI = DoctestUserInterface(config['UI'])
sage: trac = DoctestTracInterface(config['trac'], UI, DoctestTracServer())
sage: n = trac.create_ticket('Summary', 'Description', {'type':'defect', 'component':'algebr
sage: trac._get_attributes(n)['status']
'new'
sage: trac.set_attributes(n, status='needs_review')
sage: trac._get_attributes(n)['status']
'needs_review'
```

Some error checking is done:

sage: trac.set_attributes(n, status='invalid_status') Traceback (most recent call last): ... TicketSyntaxError: "invalid_status" is not a valid value for the field "status"

show_comments(*ticket*, *ignore_git_user=True*)

Shows the comments on a given ticket.

INPUT:

- `ticket` – the ticket number
- `ignore_git_user` – whether to remove comments automatically added when the branch is updated.

EXAMPLES:

```
sage: dev.trac.show_comments(100) # optional: internet
====================
was (6 years ago)
fixed
```

show_ticket(*ticket*)

Show the important fields of the given ticket.

See Also:

_get_attributes() show_comments()

EXAMPLES:

```
sage: from sage.dev.test.sagedev import single_user_setup_with_internet
sage: dev = single_user_setup_with_internet()[0]
sage: dev.trac.show_ticket(101) # optional: internet
#101: closed enhancement
== graph theory -- create a graph theory package for SAGE ==
Opened: ... years ago
Closed: ... years ago
Priority: major
Milestone: sage-2.8.5
Component: combinatorics
----------------------------------------
See http://sage.math.washington.edu:9001/graph for
initial research that Robert Miller and Emily Kirkman are doing on this.
```

CHAPTER

FOUR

USER INTERFACE

This module provides an abstract base class that is used for displaying messages and prompting for user input.

See Also:

`CmdLineInterface` for an implementation of this class

AUTHORS:

- David Roe, Julian Rueth: initial version

class `sage.dev.user_interface.`**UserInterface**(*config*)
Bases: `object`

An abstract base class for displaying messages and prompting the user for input.

TESTS:

```
sage: from sage.dev.user_interface import UserInterface
sage: from sage.dev.test.config import DoctestConfig
sage: UserInterface(DoctestConfig())
<sage.dev.user_interface.UserInterface object at 0x...>
```

confirm(*question*, *default=None*)
Ask a yes/no question and return the response as a boolean.

INPUT:

- `question` – a string
- `default` – a boolean or `None` (default: `None`), the default value

TESTS:

```
sage: from sage.dev.user_interface import UserInterface
sage: from sage.dev.test.config import DoctestConfig
sage: UI = UserInterface(DoctestConfig())
sage: UI.confirm("Should I delete your home directory?")
Traceback (most recent call last):
...
NotImplementedError
```

debug(*message*, **args*)
Display `message`.

INPUT:

- `message` – a string or list/tuple/iterable of strings.
- `*args` – optional positional arguments that will be passed to `message.format()`.

TESTS:

Debug messages are not displayed in doctests:

```
sage: from sage.dev.user_interface import UserInterface
sage: from sage.dev.test.config import DoctestConfig
sage: UI = UserInterface(DoctestConfig())
sage: UI.debug("I ate filet mignon for dinner.")
```

edit (*filename*)

Drop user into an editor with `filename` open.

OUTPUT:

Raises a `sage.dev.user_interface_error.OperationCancelledError` if the editor exits with non-zero exit code.

TESTS:

```
sage: from sage.dev.user_interface import UserInterface
sage: from sage.dev.test.config import DoctestConfig
sage: UI = UserInterface(DoctestConfig())
sage: UI.edit("filename")
Traceback (most recent call last):
...
NotImplementedError
```

error (*message*, **args*)

Display `message`.

INPUT:

- `message` – a string or list/tuple/iterable of strings.
- `*args` – optional positional arguments that will be passed to `message.format()`.

TESTS:

sage: from sage.dev.user_interface import UserInterface sage: from sage.dev.test.config import DoctestConfig sage: UI = UserInterface(DoctestConfig()) sage: UI.error("I ate filet mignon for dinner.") Traceback (most recent call last): ... NotImplementedError

get_input (*prompt*)

Read input after displaying `prompt`.

TESTS:

```
sage: from sage.dev.user_interface import UserInterface
sage: from sage.dev.test.config import DoctestConfig
sage: UI = UserInterface(DoctestConfig())
sage: UI.get_input("What do you want for dinner?")
Traceback (most recent call last):
...
NotImplementedError
```

get_password (*prompt*)

Ask for a password after displaying `prompt`.

TESTS:

```
sage: from sage.dev.user_interface import UserInterface
sage: from sage.dev.test.config import DoctestConfig
sage: UI = UserInterface(DoctestConfig())
sage: UI.get_password("What is the passphrase for your safe?")
```

```
    Traceback (most recent call last):
    ...
    NotImplementedError
```

info(*message*, **args*)

Display `message`.

These are informational messages to be shown to the user, typically indicating how to proceed.

INPUT:

- `message` – a string or list/tuple/iterable of strings.
- `*args` – optional positional arguments that will be passed to `message.format()`.

TESTS:

Info messages are not displayed in doctests:

```
sage: from sage.dev.user_interface import UserInterface
sage: from sage.dev.test.config import DoctestConfig
sage: UI = UserInterface(DoctestConfig())
sage: UI.info("I ate filet mignon for dinner.")
Traceback (most recent call last):
...
NotImplementedError
```

select(*prompt*, *options*, *default=None*)

Ask the user to select from list of options and return selected option.

INPUT:

- `prompt` – a string, prompt to display
- `options` – iterable of strings, the options
- `default` – an integer or `None` (default: `None`), the index of the default option

TESTS:

```
sage: from sage.dev.user_interface import UserInterface
sage: from sage.dev.test.config import DoctestConfig
sage: UI = UserInterface(DoctestConfig())
sage: UI.select("Should I delete your home directory?", ("yes","no","maybe"), default=1)
Traceback (most recent call last):
...
NotImplementedError
```

show(*message*, **args*, ***kwds*)

Display `message`.

INPUT:

- `message` – a string or list/tuple/iterable of strings. Each individual message will be wrapped to the terminal width.
- `*args` – optional positional arguments that will be passed to `message.format()`.
- `log_level=INFO` – one of `ERROR`, `WARNING`, `NORMAL`, `INFO`, or `DEBUG` (default: `INFO`). Optional keyword argument.

TESTS:

```
sage: from sage.dev.user_interface import UserInterface, DEBUG
sage: from sage.dev.test.config import DoctestConfig
sage: UI = UserInterface(DoctestConfig())
sage: UI.show("I ate filet mignon for dinner.")
Traceback (most recent call last):
...
NotImplementedError
sage: UI.show("I ate filet mignon for dinner.", log_level=DEBUG)
```

warning(*message*, **args*)

Display `message`.

INPUT:

- `message` – a string or list/tuple/iterable of strings.
- `*args` – optional positional arguments that will be passed to `message.format()`.

TESTS:

sage: from sage.dev.user_interface import UserInterface sage: from sage.dev.test.config import DoctestConfig sage: UI = UserInterface(DoctestConfig()) sage: UI.warning("I ate filet mignon for dinner.") Traceback (most recent call last): ... NotImplementedError

CHAPTER

FIVE

INDICES AND TABLES

- Index
- Module Index
- Search Page

PYTHON MODULE INDEX

d

INDEX

D

E

F

G

H

I

L

M

N

P

Q

R

S

T

U

V

W