

# Python Application Building & Version Control



AY 250  
Isaac Shivvers  
Sep., 2012

# Important reminders

Email us at:

[ucbpythonclass+seminar@gmail.com](mailto:ucbpythonclass+seminar@gmail.com)

Keep updated with the git repository:

```
git clone http://github.com/python-seminar.git
{-- and often do a pull --}
git pull
```

We'll talk a lot more about git later today

If you want the bootcamp slides &etc, find them at: <http://www.pythonbootcamp.info/agenda>

Friday help sessions will now be 11am - 1pm,  
since no-one really wants to come in at 10  
anyway

# Outline

- Managing Packages -  
`pip, setup.py, virtualenv`
- Command Line Parsing - `argparse`
- Building Modules & Packages
  - <breakout session>
- Version Control - `git`
- Debugging & Testing - `pdb, ipy %debug, pylint, pep8`
- Distribution - `distutils2`

# Getting Packages - `setup.py`

Installing a package with `setup.py`

# Getting Packages - `setup.py`

- The most straightforward way to get packages is to download them from the developer's website, if they've followed the standard conventions

Installing a package with `setup.py`

# Getting Packages - `setup.py`

- The most straightforward way to get packages is to download them from the developer's website, if they've followed the standard conventions
- There is a standard Python package distribution scheme using `distutils2` and `setup.py` files

Installing a package with `setup.py`

# Getting Packages - `setup.py`

- The most straightforward way to get packages is to download them from the developer's website, if they've followed the standard conventions
- There is a standard Python package distribution scheme using `distutils2` and `setup.py` files
  - stay tuned...

Installing a package with `setup.py`

# Getting Packages - `setup.py`

- The most straightforward way to get packages is to download them from the developer's website, if they've followed the standard conventions
- There is a standard Python package distribution scheme using `distutils2` and `setup.py` files
  - stay tuned...

## Installing a package with `setup.py`

```
$ cd [folder with package and setup.py file]
$ sudo python setup.py install
[ progress report ... ]
$ Finished processing dependencies for [package]

{-- if you want more info, there are several options to modify --}
$ python setup.py --help install
```

# Getting Packages - pip

# Getting Packages - pip

- `pip` is a package manager for Python, similar to apt-get for Ubuntu or MacPorts/Homebrew for OsX

# Getting Packages - pip

- `pip` is a package manager for Python, similar to apt-get for Ubuntu or MacPorts/Homebrew for OsX
- `pip --> Pip Installs Packages`

# Getting Packages - pip

- `pip` is a package manager for Python, similar to apt-get for Ubuntu or MacPorts/Homebrew for OsX
- `pip --> Pip Installs Packages`
- `easy_install` is the outdated version - still works, but is being phased out

# Getting Packages - pip

- `pip` is a package manager for Python, similar to apt-get for Ubuntu or MacPorts/Homebrew for OsX
- `pip --> Pip Installs Packages`
- `easy_install` is the outdated version - still works, but is being phased out
- Run from the command line (not within Python), it is automatically associated with your python installation

# Getting Packages - pip

- `pip` is a package manager for Python, similar to apt-get for Ubuntu or MacPorts/Homebrew for OsX
  - `pip` --> Pip Installs Packages
  - `easy_install` is the outdated version - still works, but is being phased out
  - Run from the command line (not within Python), it is automatically associated with your python installation
  - Downloads packages from the official PyPI - the Python Package Index

# Getting Packages - pip

- `pip` is a package manager for Python, similar to apt-get for Ubuntu or MacPorts/Homebrew for OsX
  - `pip --> Pip Installs Packages`
  - `easy_install` is the outdated version - still works, but is being phased out
  - Run from the command line (not within Python), it is automatically associated with your python installation
  - Downloads packages from the official PyPI - the Python Package Index
  - May have to install `pip` by downloading it, and using its `setup.py` file

# Getting Packages - pip

Installing a package

Upgrading a package

Uninstalling a package

# Getting Packages - pip

## Installing a package

```
$ sudo pip install simplejson  
[... progress report ...]  
Successfully installed simplejson
```

## Upgrading a package

## Uninstalling a package

# Getting Packages - pip

## Installing a package

```
$ sudo pip install simplejson  
[... progress report ...]  
Successfully installed simplejson
```

## Upgrading a package

```
$ sudo pip install --upgrade simplejson  
[... progress report ...]  
Successfully installed simplejson
```

## Uninstalling a package

# Getting Packages - pip

## Installing a package

```
$ sudo pip install simplejson  
[... progress report ...]  
Successfully installed simplejson
```

## Upgrading a package

```
$ sudo pip install --upgrade simplejson  
[... progress report ...]  
Successfully installed simplejson
```

## Uninstalling a package

```
$ sudo pip uninstall simplejson  
Uninstalling simplejson:  
 /home/me/env/lib/python2.7/site-packages/simplejson  
 /home/me/env/lib/python2.7/site-packages/simplejson-2.2.1-  
 py2.7.egg-info  
$ Proceed (y/n)? y  
 Successfully uninstalled simplejson
```

# Getting Packages - pip

What if you don't have superuser privileges? E.G. on a department computer. You can install packages to your own folder, and include them by modifying your .bashrc or .profile file.

with PIP

with setup.py

Add to your .bashrc, .cshrc, or .tcshrc file:

BASH Style: `export PYTHONPATH=/path/to/my_choice`

CSH Style: `setenv PYTHONPATH /path/to/my_choice`

# Getting Packages - pip

What if you don't have superuser privileges? E.G. on a department computer. You can install packages to your own folder, and include them by modifying your .bashrc or .profile file.

with PIP

```
$ sudo pip install simplejson --target=<my_choice>
```

with setup.py

Add to your .bashrc, .cshrc, or .tcshrc file:

BASH Style: export PYTHONPATH=/path/to/my\_choice

CSH Style: setenv PYTHONPATH /path/to/my\_choice

# Getting Packages - pip

What if you don't have superuser privileges? E.G. on a department computer. You can install packages to your own folder, and include them by modifying your .bashrc or .profile file.

with PIP

```
$ sudo pip install simplejson --target=<my_choice>
```

with setup.py

```
{-- on unix --}  
$ python setup.py install --home <my_choice>  
  
{-- on windows --}  
$ python setup.py install --prefix "my_choice"
```

Add to your .bashrc, .cshrc, or .tcshrc file:

BASH Style: export PYTHONPATH=/path/to/my\_choice

CSH Style: setenv PYTHONPATH /path/to/my\_choice

# Managing Packages - virtualenv

# Managing Packages - `virtualenv`

- Open Source software is constantly changing - how do you protect working code against future updates?

# Managing Packages - `virtualenv`

- Open Source software is constantly changing - how do you protect working code against future updates?
- Or, what if there is a beta release of a package you want to try, but you don't want to fully commit yet?

# Managing Packages - `virtualenv`

- Open Source software is constantly changing - how do you protect working code against future updates?
- Or, what if there is a beta release of a package you want to try, but you don't want to fully commit yet?
- `virtualenv` creates a local, self-contained, and totally separate python installation.

# Managing Packages - `virtualenv`

- Open Source software is constantly changing - how do you protect working code against future updates?
- Or, what if there is a beta release of a package you want to try, but you don't want to fully commit yet?
- `virtualenv` creates a local, self-contained, and totally separate python installation.
- Use it to create a local Python ecosystem, separate from your computer's main system, so that you can do what you want in one without affecting the other.

# virtualenv

Installing

Creating a new environment

Creating a new environment, including packages installed system-wide:

# virtualenv

## Installing

```
$ sudo pip install virtualenv  
[ progress report ... ]  
Successfully installed virtualenv  
Cleaning up...
```

## Creating a new environment

Creating a new environment, including packages installed system-wide:

# virtualenv

## Installing

```
$ sudo pip install virtualenv  
[ progress report ... ]  
Successfully installed virtualenv  
Cleaning up...
```

## Creating a new environment

```
$ virtualenv LocalPython  
New python executable in LocalPython/bin/python  
Installing setuptools.....done.  
Installing pip.....done.
```

Creating a new environment, including packages installed system-wide:

# virtualenv

## Installing

```
$ sudo pip install virtualenv  
[ progress report ... ]  
Successfully installed virtualenv  
Cleaning up...
```

## Creating a new environment

```
$ virtualenv LocalPython  
New python executable in LocalPython/bin/python  
Installing setuptools.....done.  
Installing pip.....done.
```

## Creating a new environment, including packages installed system-wide:

```
$ virtualenv LocalPython --system-site-packages
```

# virtualenv

New python executable is created

You can open a script with this environment,  
or run the interpreter, by calling it directly:

# virtualenv

New python executable is created

You can open a script with this environment,  
or run the interpreter, by calling it directly:

```
$ LocalPython/bin/python my_script.py  
[ OR ]  
  
$ LocalPython/bin/python  
Python 2.7.2 (default, Jun 20 2012, 16:23:33)  
[GCC 4.2.1 Compatible Apple Clang 4.0 (tags/Apple/  
clang-418.0.60)] on darwin  
Type "help", "copyright", "credits" or "license" for more  
information.  
>>>
```

# virtualenv

New python executable is created

```
$ ls LocalPython/bin/  
activate          activate_this.py  pip           python2.7  
activate.csh      easy_install    pip-2.7  
activate.fish     easy_install-2.7  python
```

You can open a script with this environment,  
or run the interpreter, by calling it directly:

```
$ LocalPython/bin/python my_script.py  
  
[ OR ]  
  
$ LocalPython/bin/python  
Python 2.7.2 (default, Jun 20 2012, 16:23:33)  
[GCC 4.2.1 Compatible Apple Clang 4.0 (tags/Apple/  
clang-418.0.60)] on darwin  
Type "help", "copyright", "credits" or "license" for more  
information.  
>>>
```

# virtualenv

During a shell session, you can source this environment so that it runs as the default:

Run the command 'deactivate' to exit:

Just delete to remove environment:

# virtualenv

During a shell session, you can source this environment so that it runs as the default:

```
$ source LocalPython/bin/activate  
(LocalPython)$  
[ pip or python now point to new environment ]  
(LocalPython)$ which python  
LocalPython/bin/python
```

Run the command 'deactivate' to exit:

Just delete to remove environment:

# virtualenv

During a shell session, you can source this environment so that it runs as the default:

```
$ source LocalPython/bin/activate  
(LocalPython)$  
[ pip or python now point to new environment ]  
(LocalPython)$ which python  
LocalPython/bin/python
```

Run the command `deactivate` to exit:

```
(LocalPython)$ deactivate  
$ which python  
/usr/bin/python
```

Just delete to remove environment:

# virtualenv

During a shell session, you can source this environment so that it runs as the default:

```
$ source LocalPython/bin/activate  
(LocalPython)$  
[ pip or python now point to new environment ]  
(LocalPython)$ which python  
LocalPython/bin/python
```

Run the command `deactivate` to exit:

```
(LocalPython)$ deactivate  
$ which python  
/usr/bin/python
```

Just delete to remove environment:

```
$ rm -r LocalPython
```

# Outline

- Managing Packages -  
`pip, setup.py, virtualenv`
- Command Line Parsing - `argparse`
- Building Modules & Packages  
  
`<breakout session>`
- Version Control - `git`
- Debugging & Testing - `pdb, ipy %debug, pylint, pep8`
- Distribution - `distutils2`

# Command Line Parsing

```
$ python myawesomeprogram.py -o option1 -p parameter2 -Q -R
```

# Command Line Parsing

```
$ python myawesomeprogram.py -o option1 -p parameter2 -Q -R
```

- **Goal:** build a command-line 'standalone' codebase in Python, w/ CL options & keywords

# Command Line Parsing

```
$ python myawesomeprogram.py -o option1 -p parameter2 -Q -R
```

- **Goal:** build a command-line 'standalone' codebase in Python, w/ CL options & keywords
- **Solution:** [argparse](#), which has been built in to Python 2.7 & above (if you don't have it, you can get it with 'pip argparse')

# Command Line Parsing

```
$ python myawesomeprogram.py -o option1 -p parameter2 -Q -R
```

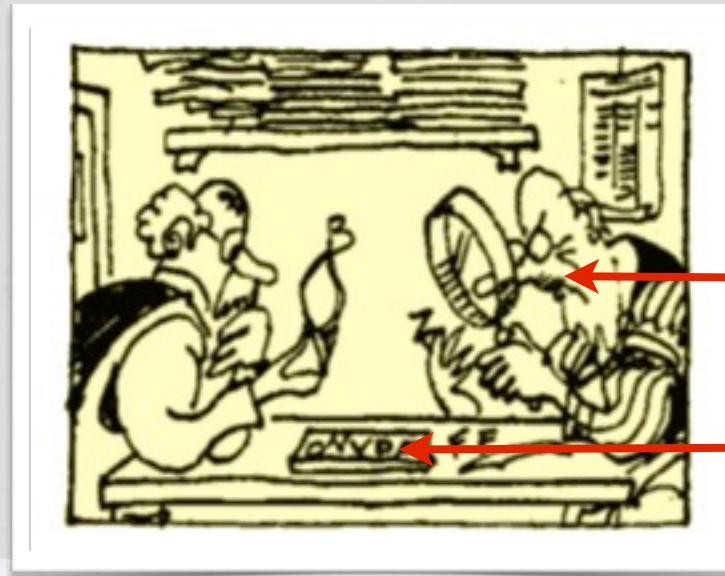
- **Goal:** build a command-line 'standalone' codebase in Python, w/ CL options & keywords
- **Solution:** [argparse](#), which has been built in to Python 2.7 & above (if you don't have it, you can get it with 'pip argparse')
- Allows for user-friendly command line interfaces, and leaves it up to the code to determine what it was the user wanted.

# Command Line Parsing

```
$ python myawesomeprogram.py -o option1 -p parameter2 -Q -R
```

- **Goal:** build a command-line 'standalone' codebase in Python, w/ CL options & keywords
- **Solution:** [argparse](#), which has been built in to Python 2.7 & above (if you don't have it, you can get it with 'pip argparse')
- Allows for user-friendly command line interfaces, and leaves it up to the code to determine what it was the user wanted.
- Also automatically generates help & usage messages and issues errors when invalid arguments are provided

# Note on optparse

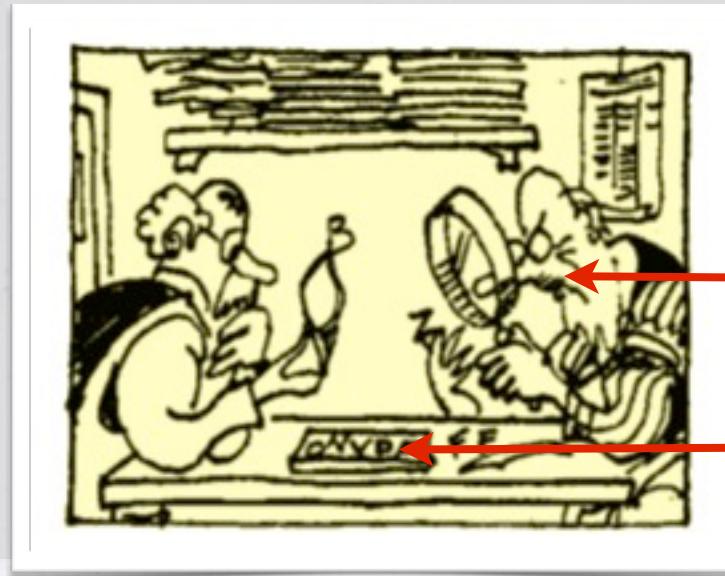


argparse

user input

# Note on optparse

- `optparse` did a similar thing, though it is being replaced in favor of `argparse`

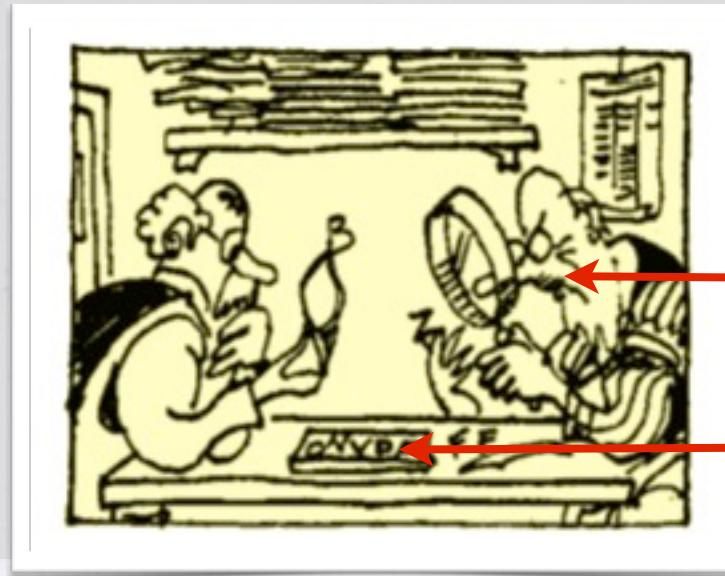


`argparse`

`user input`

# Note on optparse

- `optparse` did a similar thing, though it is being replaced in favor of `argparse`
- Not 100% compatible, but `optparse` folks should consider switching

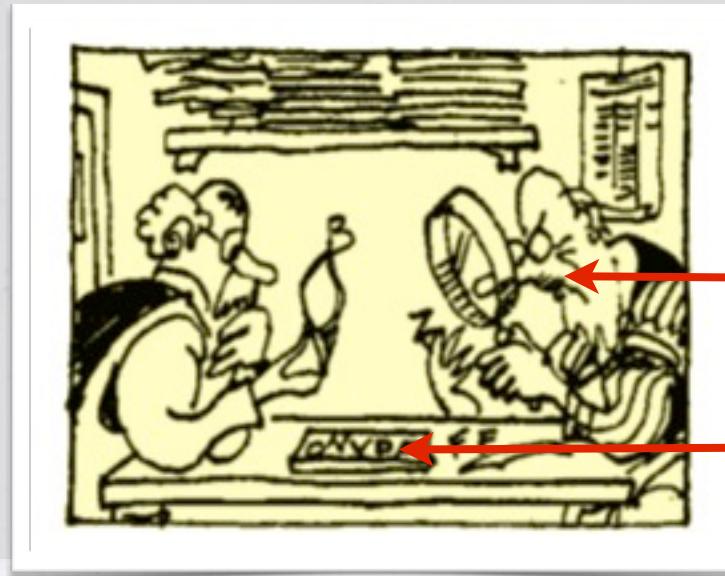


`argparse`

user input

# Note on optparse

- `optparse` did a similar thing, though it is being replaced in favor of `argparse`
- Not 100% compatible, but `optparse` folks should consider switching
- New users to either of them should use `argparse`, if it is available on your platform.



`argparse`

`user input`

# Setting up a parser

<http://www.doughellmann.com/PyMOTW/argparse/>

# Setting up a parser

- First step for `argparse`: create parser object & tell it what arguments to expect.

<http://www.doughellmann.com/PyMOTW/argparse/>

# Setting up a parser

- First step for `argparse`: create parser object & tell it what arguments to expect.
- It can then be used to process the command line arguments on runtime

<http://www.doughellmann.com/PyMOTW/argparse/>

# Setting up a parser

- First step for `argparse`: create parser object & tell it what arguments to expect.
- It can then be used to process the command line arguments on runtime
- Parser class: `ArgumentParser`. Takes several arguments to set up the description used in the help text for the program & other global behaviors

<http://www.doughellmann.com/PyMOTW/argparse/>

# Setting up a parser

- First step for `argparse`: create parser object & tell it what arguments to expect.
- It can then be used to process the command line arguments on runtime
- Parser class: `ArgumentParser`. Takes several arguments to set up the description used in the help text for the program & other global behaviors

```
import argparse
```

<http://www.doughellmann.com/PyMOTW/argparse/>

# Setting up a parser

- First step for `argparse`: create parser object & tell it what arguments to expect.
- It can then be used to process the command line arguments on runtime
- Parser class: `ArgumentParser`. Takes several arguments to set up the description used in the help text for the program & other global behaviors

```
import argparse  
parser = argparse.ArgumentParser(description='Sample Application')
```

<http://www.doughellmann.com/PyMOTW/argparse/>

# Defining Arguments & Parsing

<http://www.doughellmann.com/PyMOTW/argparse/>

# Defining Arguments & Parsing

- Arguments can trigger different actions, specified by the action argument to `add_argument()`.

<http://www.doughellmann.com/PyMOTW/argparse/>

# Defining Arguments & Parsing

- Arguments can trigger different actions, specified by the action argument to `add_argument()`.
- Several supported actions (next slide).

<http://www.doughellmann.com/PyMOTW/argparse/>

# Defining Arguments & Parsing

- Arguments can trigger different actions, specified by the action argument to `add_argument()`.
- Several supported actions (next slide).
- Once all of the arguments are defined, you can parse the command line by passing a sequence of argument strings to `parse_args()`.

<http://www.doughellmann.com/PyMOTW/argparse/>

# Defining Arguments & Parsing

- Arguments can trigger different actions, specified by the action argument to `add_argument()`.
- Several supported actions (next slide).
- Once all of the arguments are defined, you can parse the command line by passing a sequence of argument strings to `parse_args()`.
- By default, arguments are taken from `sys.argv[1:]`, but you can also pass your own list.

<http://www.doughellmann.com/PyMOTW/argparse/>

# Argument Actions

h/t <http://www.doughellmann.com/PyMOTW/argparse/>

# Argument Actions

- **store**: Save the value, after optionally converting it to a different type (default)

h/t <http://www.doughellmann.com/PyMOTW/argparse/>

# Argument Actions

- `store`: Save the value, after optionally converting it to a different type (default)
- `store_const`: Save the value as defined as part of the argument specification, rather than a value that comes from the arguments being parsed

h/t <http://www.doughellmann.com/PyMOTW/argparse/>

# Argument Actions

- `store`: Save the value, after optionally converting it to a different type (default)
- `store_const`: Save the value as defined as part of the argument specification, rather than a value that comes from the arguments being parsed
- `store_true/store_false`: Save the appropriate boolean value

h/t <http://www.doughellmann.com/PyMOTW/argparse/>

# Argument Actions

- `store`: Save the value, after optionally converting it to a different type (default)
- `store_const`: Save the value as defined as part of the argument specification, rather than a value that comes from the arguments being parsed
- `store_true/store_false`: Save the appropriate boolean value
- `append`: Save the value to a list. Multiple values are saved if the argument is repeated

h/t <http://www.doughellmann.com/PyMOTW/argparse/>

# Argument Actions

- `store`: Save the value, after optionally converting it to a different type (default)
- `store_const`: Save the value as defined as part of the argument specification, rather than a value that comes from the arguments being parsed
- `store_true/store_false`: Save the appropriate boolean value
- `append`: Save the value to a list. Multiple values are saved if the argument is repeated
- `append_const`: Save a value defined in the argument specification to a list

h/t <http://www.doughellmann.com/PyMOTW/argparse/>

# Argument Actions

- **store**: Save the value, after optionally converting it to a different type (default)
- **store\_const**: Save the value as defined as part of the argument specification, rather than a value that comes from the arguments being parsed
- **store\_true/store\_false**: Save the appropriate boolean value
- **append**: Save the value to a list. Multiple values are saved if the argument is repeated
- **append\_const**: Save a value defined in the argument specification to a list
- **version**: Prints version details about the program and then exits

h/t <http://www.doughellmann.com/PyMOTW/argparse/>

# Argument Actions

```
import argparse
parser = argparse.ArgumentParser(description='Sample Application')
parser.add_argument('-s', action='store', dest='simple_value',
                    help='Store a simple value')
parser.add_argument('-c', action='store_const', dest='constant_value',
                    const='value-to-store',
                    help='Store a constant value')
parser.add_argument('-t', action='store_true', default=False,
                    dest='boolean_switch',
                    help='Set a switch to true')
parser.add_argument('-f', action='store_false', default=False,
                    dest='boolean_switch',
                    help='Set a switch to false')
parser.add_argument('-a', action='append', dest='collection',
                    default=[],
                    help='Add repeated values to a list',
                    )
parser.add_argument('-A', action='append_const', dest='const_collection',
                    const='value-1-to-append',
                    default=[],
                    help='Add different values to list')
parser.add_argument('-B', action='append_const', dest='const_collection',
                    const='value-2-to-append',
                    help='Add different values to list')
parser.add_argument('--version', action='version', version='%(prog)s 1.0')
results = parser.parse_args()
print 'simple_value      =', results.simple_value
print 'constant_value    =', results.constant_value
print 'boolean_switch    =', results.boolean_switch
print 'collection        =', results.collection
print 'const_collection  =', results.const_colle
```

h/t <http://www.doughellmann.com/PyMOTW/argparse/>

# Argument Actions - example

help

<http://www.doughellmann.com/PyMOTW/argparse/>

# Argument Actions - example

help

```
$ python argparse_action.py -h

usage: argparse_action.py [-h] [-s SIMPLE_VALUE] [-c] [-t] [-f]
                          [-a COLLECTION] [-A] [-B] [--version]

optional arguments:
  -h, --help            show this help message and exit
  -s SIMPLE_VALUE      Store a simple value
  -c                  Store a constant value
  -t                  Set a switch to true
  -f                  Set a switch to false
  -a COLLECTION        Add repeated values to a list
  -A                  Add different values to list
  -B                  Add different values to list
  --version           show program's version number and exit
```

<http://www.doughellmann.com/PyMOTW/argparse/>

# Argument Actions - example

store

```
parser.add_argument('-s', action='store', dest='simple_value',
                    help='Store a simple value')
```

<http://www.doughellmann.com/PyMOTW/argparse/>

# Argument Actions - example

store

```
parser.add_argument('-s', action='store', dest='simple_value',
                    help='Store a simple value')
```

```
$ python argparse_action.py -s value
```

<http://www.doughellmann.com/PyMOTW/argparse/>

# Argument Actions - example

store

```
parser.add_argument('-s', action='store', dest='simple_value',
                    help='Store a simple value')
```

```
$ python argparse_action.py -s value
simple_value      = value
```

<http://www.doughellmann.com/PyMOTW/argparse/>

# Argument Actions - example

store

```
parser.add_argument('-s', action='store', dest='simple_value',
                    help='Store a simple value')
```

```
$ python argparse_action.py -s value
simple_value      = value
constant_value    = None
```

<http://www.doughellmann.com/PyMOTW/argparse/>

# Argument Actions - example

store

```
parser.add_argument('-s', action='store', dest='simple_value',
                    help='Store a simple value')
```

```
$ python argparse_action.py -s value
simple_value      = value
constant_value    = None
boolean_switch    = False
```

<http://www.doughellmann.com/PyMOTW/argparse/>

# Argument Actions - example

store

```
parser.add_argument('-s', action='store', dest='simple_value',
                    help='Store a simple value')
```

```
$ python argparse_action.py -s value
simple_value      = value
constant_value    = None
boolean_switch   = False
collection        = [ ]
```

<http://www.doughellmann.com/PyMOTW/argparse/>

# Argument Actions - example

store

```
parser.add_argument('-s', action='store', dest='simple_value',
                    help='Store a simple value')
```

```
$ python argparse_action.py -s value
simple_value      = value
constant_value    = None
boolean_switch   = False
collection        = []
const_collection = []
```

<http://www.doughellmann.com/PyMOTW/argparse/>

# Argument Actions - example

## `store_const`

```
parser.add_argument('-c', action='store_const', dest='constant_value',
                    const='value-to-store',
                    help='Store a constant value')
```

<http://www.doughellmann.com/PyMOTW/argparse/>

# Argument Actions - example

## `store_const`

```
parser.add_argument('-c', action='store_const', dest='constant_value',
                    const='value-to-store',
                    help='Store a constant value')
```

```
$ python argparse_action.py -c
```

<http://www.doughellmann.com/PyMOTW/argparse/>

# Argument Actions - example

## `store_const`

```
parser.add_argument('-c', action='store_const', dest='constant_value',
                    const='value-to-store',
                    help='Store a constant value')
```

```
$ python argparse_action.py -c
simple_value      = None
```

<http://www.doughellmann.com/PyMOTW/argparse/>

# Argument Actions - example

## `store_const`

```
parser.add_argument('-c', action='store_const', dest='constant_value',
                    const='value-to-store',
                    help='Store a constant value')
```

```
$ python argparse_action.py -c
simple_value      = None
constant_value    = value-to-store
```

<http://www.doughellmann.com/PyMOTW/argparse/>

# Argument Actions - example

## `store_const`

```
parser.add_argument('-c', action='store_const', dest='constant_value',
                    const='value-to-store',
                    help='Store a constant value')
```

```
$ python argparse_action.py -c

simple_value      = None
constant_value    = value-to-store
boolean_switch   = False
```

<http://www.doughellmann.com/PyMOTW/argparse/>

# Argument Actions - example

## `store_const`

```
parser.add_argument('-c', action='store_const', dest='constant_value',
                    const='value-to-store',
                    help='Store a constant value')
```

```
$ python argparse_action.py -c

simple_value      = None
constant_value    = value-to-store
boolean_switch   = False
collection        = [ ]
```

<http://www.doughellmann.com/PyMOTW/argparse/>

# Argument Actions - example

## `store_const`

```
parser.add_argument('-c', action='store_const', dest='constant_value',
                    const='value-to-store',
                    help='Store a constant value')
```

```
$ python argparse_action.py -c

simple_value      = None
constant_value    = value-to-store
boolean_switch   = False
collection        = []
const_collection = []
```

<http://www.doughellmann.com/PyMOTW/argparse/>

# Argument Actions - example

## store\_true/store\_false

```
parser.add_argument('-t', action='store_true', default=False,  
                    dest='boolean_switch',  
                    help='Set a switch to true')
```

<http://www.doughellmann.com/PyMOTW/argparse/>

# Argument Actions - example

## store\_true/store\_false

```
parser.add_argument('-t', action='store_true', default=False,  
                    dest='boolean_switch',  
                    help='Set a switch to true')
```

```
$ python argparse_action.py -t
```

<http://www.doughellmann.com/PyMOTW/argparse/>

# Argument Actions - example

## store\_true/store\_false

```
parser.add_argument('-t', action='store_true', default=False,  
                    dest='boolean_switch',  
                    help='Set a switch to true')
```

```
$ python argparse_action.py -t  
simple_value      = None
```

<http://www.doughellmann.com/PyMOTW/argparse/>

# Argument Actions - example

## store\_true/store\_false

```
parser.add_argument('-t', action='store_true', default=False,  
                    dest='boolean_switch',  
                    help='Set a switch to true')
```

```
$ python argparse_action.py -t  
  
simple_value      = None  
constant_value    = None
```

<http://www.doughellmann.com/PyMOTW/argparse/>

# Argument Actions - example

## store\_true/store\_false

```
parser.add_argument('-t', action='store_true', default=False,  
                    dest='boolean_switch',  
                    help='Set a switch to true')
```

```
$ python argparse_action.py -t  
  
simple_value      = None  
constant_value    = None  
boolean_switch    = True
```

<http://www.doughellmann.com/PyMOTW/argparse/>

# Argument Actions - example

## store\_true/store\_false

```
parser.add_argument('-t', action='store_true', default=False,  
                    dest='boolean_switch',  
                    help='Set a switch to true')
```

```
$ python argparse_action.py -t  
  
simple_value      = None  
constant_value    = None  
boolean_switch    = True  
collection        = [ ]
```

<http://www.doughellmann.com/PyMOTW/argparse/>

# Argument Actions - example

## store\_true/store\_false

```
parser.add_argument('-t', action='store_true', default=False,  
                    dest='boolean_switch',  
                    help='Set a switch to true')
```

```
$ python argparse_action.py -t  
  
simple_value      = None  
constant_value    = None  
boolean_switch    = True  
collection        = []  
const_collection  = []
```

<http://www.doughellmann.com/PyMOTW/argparse/>

# Argument Actions - example

## append

```
parser.add_argument('-a', action='append', dest='collection',
                    default=[],
                    help='Add repeated values to a list',
                    )
```

<http://www.doughellmann.com/PyMOTW/argparse/>

# Argument Actions - example

## append

```
parser.add_argument('-a', action='append', dest='collection',
                    default=[],
                    help='Add repeated values to a list',
                    )
```

```
$ python argparse_action.py -a one -a two -a three
```

<http://www.doughellmann.com/PyMOTW/argparse/>

# Argument Actions - example

## append

```
parser.add_argument('-a', action='append', dest='collection',
                    default=[],
                    help='Add repeated values to a list',
                    )
```

```
$ python argparse_action.py -a one -a two -a three
simple_value      = None
```

<http://www.doughellmann.com/PyMOTW/argparse/>

# Argument Actions - example

## append

```
parser.add_argument('-a', action='append', dest='collection',
                    default=[],
                    help='Add repeated values to a list',
                    )
```

```
$ python argparse_action.py -a one -a two -a three
simple_value      = None
constant_value    = None
```

<http://www.doughellmann.com/PyMOTW/argparse/>

# Argument Actions - example

## append

```
parser.add_argument('-a', action='append', dest='collection',
                    default=[],
                    help='Add repeated values to a list',
                    )
```

```
$ python argparse_action.py -a one -a two -a three

simple_value      = None
constant_value    = None
boolean_switch   = False
```

<http://www.doughellmann.com/PyMOTW/argparse/>

# Argument Actions - example

## append

```
parser.add_argument('-a', action='append', dest='collection',
                    default=[],
                    help='Add repeated values to a list',
                    )
```

```
$ python argparse_action.py -a one -a two -a three

simple_value      = None
constant_value    = None
boolean_switch   = False
collection        = [ 'one', 'two', 'three' ]
```

<http://www.doughellmann.com/PyMOTW/argparse/>

# Argument Actions - example

## append

```
parser.add_argument('-a', action='append', dest='collection',
                    default=[],
                    help='Add repeated values to a list',
                    )
```

```
$ python argparse_action.py -a one -a two -a three

simple_value      = None
constant_value    = None
boolean_switch   = False
collection        = [ 'one', 'two', 'three' ]
const_collection = [ ]
```

<http://www.doughellmann.com/PyMOTW/argparse/>

# Argument Actions - example

## append\_const

```
parser.add_argument('-A', action='append_const', dest='const_collection',
                    const='value-1-to-append',
                    default=[],
                    help='Add different values to list')
parser.add_argument('-B', action='append_const', dest='const_collection',
                    const='value-2-to-append',
                    help='Add different values to list')
```

<http://www.doughellmann.com/PyMOTW/argparse/>

# Argument Actions - example

## append\_const

```
parser.add_argument('-A', action='append_const', dest='const_collection',
                    const='value-1-to-append',
                    default=[],
                    help='Add different values to list')
parser.add_argument('-B', action='append_const', dest='const_collection',
                    const='value-2-to-append',
                    help='Add different values to list')
```

```
$ python argparse_action.py -B -A
```

<http://www.doughellmann.com/PyMOTW/argparse/>

# Argument Actions - example

## append\_const

```
parser.add_argument('-A', action='append_const', dest='const_collection',
                    const='value-1-to-append',
                    default=[],
                    help='Add different values to list')
parser.add_argument('-B', action='append_const', dest='const_collection',
                    const='value-2-to-append',
                    help='Add different values to list')
```

```
$ python argparse_action.py -B -A
simple_value      = None
```

<http://www.doughellmann.com/PyMOTW/argparse/>

# Argument Actions - example

## append\_const

```
parser.add_argument('-A', action='append_const', dest='const_collection',
                    const='value-1-to-append',
                    default=[],
                    help='Add different values to list')
parser.add_argument('-B', action='append_const', dest='const_collection',
                    const='value-2-to-append',
                    help='Add different values to list')
```

```
$ python argparse_action.py -B -A
simple_value      = None
constant_value    = None
```

<http://www.doughellmann.com/PyMOTW/argparse/>

# Argument Actions - example

## append\_const

```
parser.add_argument('-A', action='append_const', dest='const_collection',
                    const='value-1-to-append',
                    default=[],
                    help='Add different values to list')
parser.add_argument('-B', action='append_const', dest='const_collection',
                    const='value-2-to-append',
                    help='Add different values to list')
```

```
$ python argparse_action.py -B -A
simple_value      = None
constant_value    = None
boolean_switch   = False
```

<http://www.doughellmann.com/PyMOTW/argparse/>

# Argument Actions - example

## append\_const

```
parser.add_argument('-A', action='append_const', dest='const_collection',
                    const='value-1-to-append',
                    default=[],
                    help='Add different values to list')
parser.add_argument('-B', action='append_const', dest='const_collection',
                    const='value-2-to-append',
                    help='Add different values to list')
```

```
$ python argparse_action.py -B -A
simple_value      = None
constant_value    = None
boolean_switch   = False
collection        = [ ]
```

<http://www.doughellmann.com/PyMOTW/argparse/>

# Argument Actions - example

## append\_const

```
parser.add_argument('-A', action='append_const', dest='const_collection',
                    const='value-1-to-append',
                    default=[],
                    help='Add different values to list')
parser.add_argument('-B', action='append_const', dest='const_collection',
                    const='value-2-to-append',
                    help='Add different values to list')
```

```
$ python argparse_action.py -B -A

simple_value      = None
constant_value    = None
boolean_switch   = False
collection        = []
const_collection = [ 'value-2-to-append', 'value-1-to-append' ]
```

<http://www.doughellmann.com/PyMOTW/argparse/>

# Argument Actions - example

version

```
parser.add_argument('--version', action='version', version='%(prog)s 1.0')
```

<http://www.doughellmann.com/PyMOTW/argparse/>

# Argument Actions - example

version

```
parser.add_argument('--version', action='version', version='%(prog)s 1.0')
```

```
$ python argparse_action.py --version
```

<http://www.doughellmann.com/PyMOTW/argparse/>

# Argument Actions - example

version

```
parser.add_argument('--version', action='version', version='%(prog)s 1.0')
```

```
$ python argparse_action.py --version  
argparse_action.py 1.0
```

<http://www.doughellmann.com/PyMOTW/argparse/>

# Outline

- Managing Packages -  
`pip, setup.py, virtualenv`
- Command Line Parsing - `argparse`
- Building Modules & Packages  
  
`<breakout session>`
- Version Control - `git`
- Debugging & Testing - `pdb, ipy %debug, pylint, pep8`
- Distribution - `distutils2`

# Modules and Packages



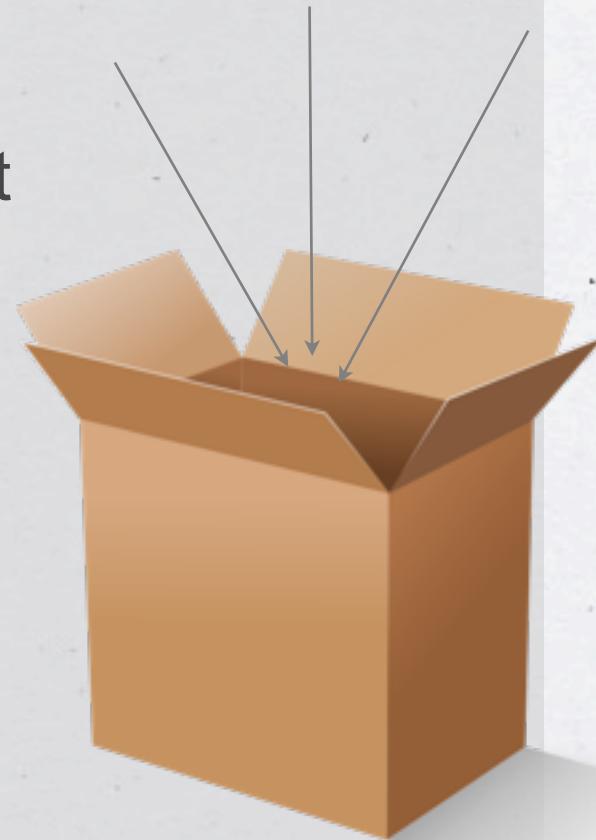
# Modules and Packages

- As code gets more involved, it becomes unwieldy & unnatural to keep everything in the same file, or even the same folder



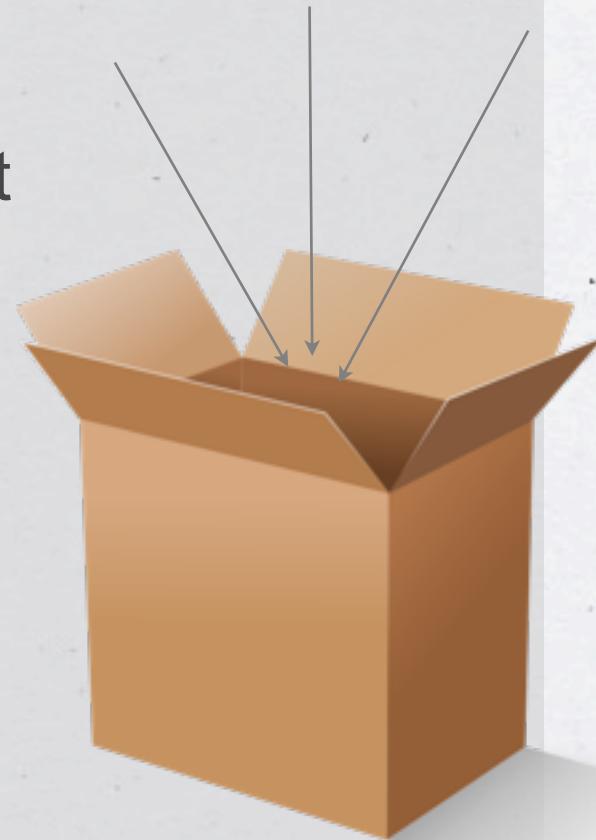
# Modules and Packages

- As code gets more involved, it becomes unwieldy & unnatural to keep everything in the same file, or even the same folder
- Functions from other codes made for different reasons might be useful elsewhere



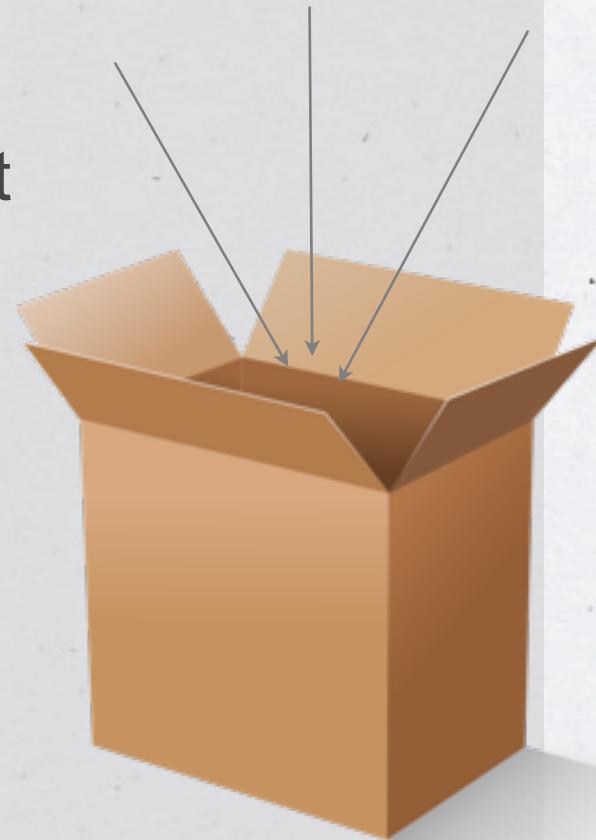
# Modules and Packages

- As code gets more involved, it becomes unwieldy & unnatural to keep everything in the same file, or even the same folder
- Functions from other codes made for different reasons might be useful elsewhere
- Useful to break up code into **modules** and **packages** - used like 'package.module'



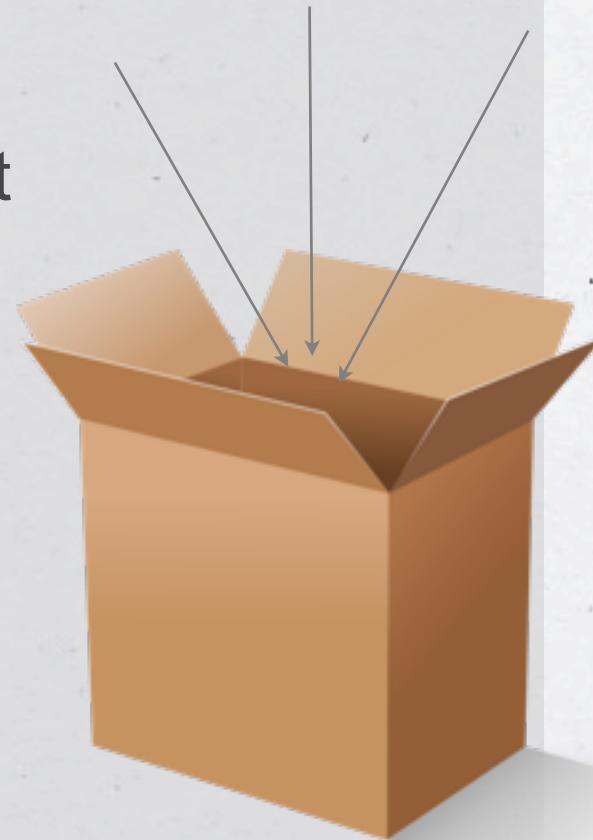
# Modules and Packages

- As code gets more involved, it becomes unwieldy & unnatural to keep everything in the same file, or even the same folder
- Functions from other codes made for different reasons might be useful elsewhere
- Useful to break up code into **modules** and **packages** - used like 'package.module'
- Module: file containing defined functions & variables. It **must have a .py extension**.



# Modules and Packages

- As code gets more involved, it becomes unwieldy & unnatural to keep everything in the same file, or even the same folder
- Functions from other codes made for different reasons might be useful elsewhere
- Useful to break up code into **modules** and **packages** - used like 'package.module'
- Module: file containing defined functions & variables. It **must have a .py extension**.
- Package: a properly-organized folder containing modules (packages Numpy are well-developed examples - you can make your own)



# Modules: Setting up your path

## PYTHONPATH

Augment the default search path for module files. The format is the same as the shell's **PATH**: one or more directory pathnames separated by `os.pathsep` (e.g. colons on Unix or semicolons on Windows). Non-existent directories are silently ignored.

In addition to normal directories, individual **PYTHONPATH** entries may refer to zipfiles containing pure Python modules (in either source or compiled form). Extension modules cannot be imported from zipfiles.

The default search path is installation dependent, but generally begins with `prefix/lib/pythonversion` (see **PYTHONHOME** above). It is *always* appended to **PYTHONPATH**.

An additional directory will be inserted in the search path in front of **PYTHONPATH** as described above under *Interface options*. The search path can be manipulated from within a Python program as the variable `sys.path`.

<http://docs.python.org/using/cmdline.html#environment-variables>

Add to your `.bashrc`, `.cshrc`, or `.tcshrc` file:

BASH Style: `export PYTHONPATH=/path/to/your/code`

CSH Style: `setenv PYTHONPATH /path/to/your/code`

# Modules: More Path Stuff

Can verify your path and append to it within python

New paths appended will *not* be preserved upon exiting python.

For long-term path appending, use **PYTHONPATH** environment variable defined in previous slide.

# Modules: More Path Stuff

Can verify your path and append to it within python

```
>>> import sys
```

New paths appended will *not* be preserved upon exiting python.

For long-term path appending, use **PYTHONPATH** environment variable defined in previous slide.

# Modules: More Path Stuff

Can verify your path and append to it within python

```
>>> import sys  
>>> # Get a list of all paths python is looking at with sys.path
```

New paths appended will *not* be preserved upon exiting python.

For long-term path appending, use **PYTHONPATH** environment variable defined in previous slide.

# Modules: More Path Stuff

Can verify your path and append to it within python

```
>>> import sys  
>>> # Get a list of all paths python is looking at with sys.path  
>>> print sys.path
```

New paths appended will *not* be preserved upon exiting python.

For long-term path appending, use **PYTHONPATH** environment variable defined in previous slide.

# Modules: More Path Stuff

Can verify your path and append to it within python

```
>>> import sys  
>>> # Get a list of all paths python is looking at with sys.path  
>>> print sys.path  
[ '', '/predefined/python/path/' , '/path/defined/by/environment/variable' ]
```

New paths appended will *not* be preserved upon exiting python.

For long-term path appending, use **PYTHONPATH** environment variable defined in previous slide.

# Modules: More Path Stuff

Can verify your path and append to it within python

```
>>> import sys  
>>> # Get a list of all paths python is looking at with sys.path  
>>> print sys.path  
[ '', '/predefined/python/path/', '/path/defined/by/environment/variable' ]  
>>> # Can append to this list:
```

New paths appended will *not* be preserved upon exiting python.

For long-term path appending, use **PYTHONPATH** environment variable defined in previous slide.

# Modules: More Path Stuff

Can verify your path and append to it within python

```
>>> import sys  
>>> # Get a list of all paths python is looking at with sys.path  
>>> print sys.path  
[ '', '/predefined/python/path/', '/path/defined/by/environment/variable' ]  
>>> # Can append to this list:  
>>> # sys.path.append("/new/software/path/")
```

New paths appended will *not* be preserved upon exiting python.

For long-term path appending, use **PYTHONPATH** environment variable defined in previous slide.

# Packages

```
analysis/  
analysis/__init__.py  
analysis/datacleaner.py  
analysis/datagenerator.py  
plotting/  
plotting/__init__.py  
plotting/histogram.py  
plotting/scatterplot.py
```

```
# import all datagenerator functions:  
import analysis.datagenerator as dg  
# import the function nicehist from  
# the plotting.histogram package  
from plotting.histogram import nicehist  
  
# dg has a generate_data function  
mydata = dg.generate_data()  
nicehist(mydata)
```

# Packages

- If path is set correctly, code can be broken up into reasonable folders and imported as necessary, either by importing entire modules (.py files) or functions/classes within the modules.

```
analysis/  
analysis/__init__.py  
analysis/datacleaner.py  
analysis/datagenerator.py  
plotting/  
plotting/__init__.py  
plotting/histogram.py  
plotting/scatterplot.py
```

```
# import all datagenerator functions:  
import analysis.datagenerator as dg  
# import the function nicehist from  
# the plotting.histogram package  
from plotting.histogram import nicehist  
  
# dg has a generate_data function  
mydata = dg.generate_data()  
nicehist(mydata)
```

# Packages

- If path is set correctly, code can be broken up into reasonable folders and imported as necessary, either by importing entire modules (.py files) or functions/classes within the modules.
- Put an `__init__.py` file in each folder you want to be able to import from.

```
analysis/  
analysis/__init__.py  
analysis/datacleaner.py  
analysis/datagenerator.py  
plotting/  
plotting/__init__.py  
plotting/histogram.py  
plotting/scatterplot.py
```

```
# import all datagenerator functions:  
import analysis.datagenerator as dg  
# import the function nicehist from  
# the plotting.histogram package  
from plotting.histogram import nicehist  
  
# dg has a generate_data function  
mydata = dg.generate_data()  
nicehist(mydata)
```

# Packages

- If path is set correctly, code can be broken up into reasonable folders and imported as necessary, either by importing entire modules (.py files) or functions/classes within the modules.
- Put an `__init__.py` file in each folder you want to be able to import from.
- Code in `__init__.py` is run when the package, or any derivative of it, is imported. Usually `__init__.py` is an empty file.

```
analysis/  
analysis/__init__.py  
analysis/datacleaner.py  
analysis/datagenerator.py  
plotting/  
plotting/__init__.py  
plotting/histogram.py  
plotting/scatterplot.py
```

```
# import all datagenerator functions:  
import analysis.datagenerator as dg  
# import the function nicehist from  
# the plotting.histogram package  
from plotting.histogram import nicehist  
  
# dg has a generate_data function  
mydata = dg.generate_data()  
nicehist(mydata)
```

# Breakout Session

- Go to the breakout folder in  
.../Breakouts/01\_Versioning\_Application\_Building/
- Work on the file breakout1.py. Do not move or modify the other files, in the other folders, but you will need to use them. (You may add files to these directories, if necessary)
- Build up a command line parser which allows the user to specify:
  - how many datapoints to generate
  - whether to plot with a filled in histogram or an outlined one
  - the title of the plot
- And then have the plot be generated.
- Want to be able to run a command like:  
`bash-3.2$ python breakout1.py -t -n 200 -T "My Awesome Title"`

# Outline

- Managing Packages -  
`pip, setup.py, virtualenv`
- Command Line Parsing - `argparse`
- Building Modules & Packages  
  
`<breakout session>`
  - Version Control - `git`
  - Debugging & Testing - `pdb, ipy %debug, pylint, pep8`
  - Distribution - `distutils2`

# Version Control (with Git)



# Video Game Analogy



# Video Game Analogy

- Very old school: No save slots



# Video Game Analogy

- Very old school: No save slots



# Video Game Analogy

# Video Game Analogy

- One save slot

# Video Game Analogy

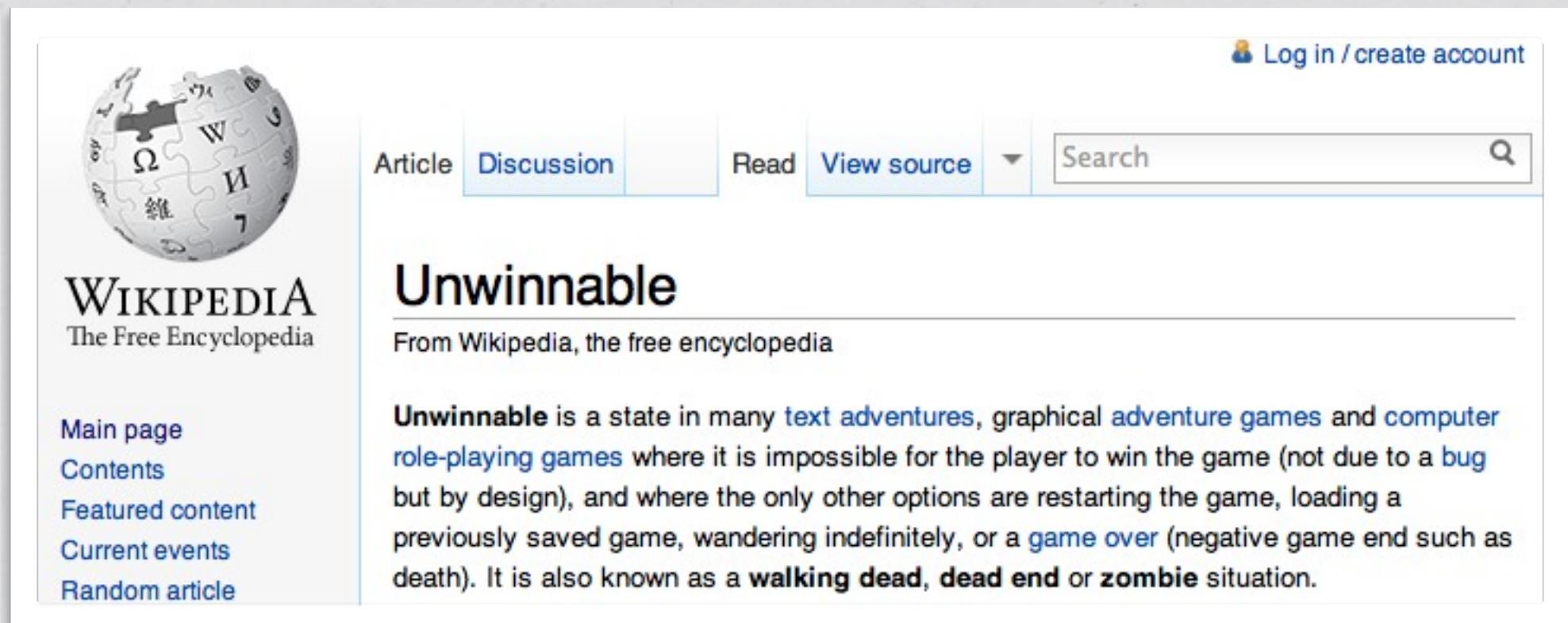
- One save slot
  - What if you want to go back to previous state?

# Video Game Analogy

- One save slot
  - What if you want to go back to previous state?
  - Worse, what if current save is unwinnable?

# Video Game Analogy

- One save slot
- What if you want to go back to previous state?
- Worse, what if current save is unwinnable?



The screenshot shows a Wikipedia page titled "Unwinnable". The page header includes the Wikipedia logo, a "Log in / create account" link, and navigation tabs for "Article", "Discussion", "Read", "View source", and a search bar. The main content area features the title "Unwinnable" in large bold letters, followed by the text "From Wikipedia, the free encyclopedia". Below this, a detailed description of the concept of an unwinnable state in games is provided.

**Unwinnable**

From Wikipedia, the free encyclopedia

**Unwinnable** is a state in many [text adventures](#), [graphical adventure games](#) and [computer role-playing games](#) where it is impossible for the player to win the game (not due to a [bug](#) but by design), and where the only other options are restarting the game, loading a previously saved game, wandering indefinitely, or a [game over](#) (negative game end such as death). It is also known as a [walking dead](#), [dead end](#) or [zombie](#) situation.

# Video Game Analogy

- One save slot
- What if you want to go back to previous state?
- Worse, what if current save is unwinnable?



The screenshot shows a Wikipedia page titled "Unwinnable". The page header includes the Wikipedia logo, a "Log in / create account" link, and navigation tabs for "Article", "Discussion", "Read", "View source", and a search bar. The main content area features the title "Unwinnable" in large bold letters, followed by a subtitle "From Wikipedia, the free encyclopedia". A large black emoticon ":(" is displayed next to the subtitle. The text of the article describes the concept of an unwinnable state in games, mentioning text adventures, graphical adventure games, and computer role-playing games. It explains that it is impossible for the player to win the game, either due to a bug or by design, and that other options like restarting, loading a previous save, or wandering indefinitely are available. The text also refers to negative game ends like "game over" and "death", and alternative names like "walking dead", "dead end", and "zombie situation".

WIKIPEDIA  
The Free Encyclopedia

Main page  
Contents  
Featured content  
Current events  
Random article

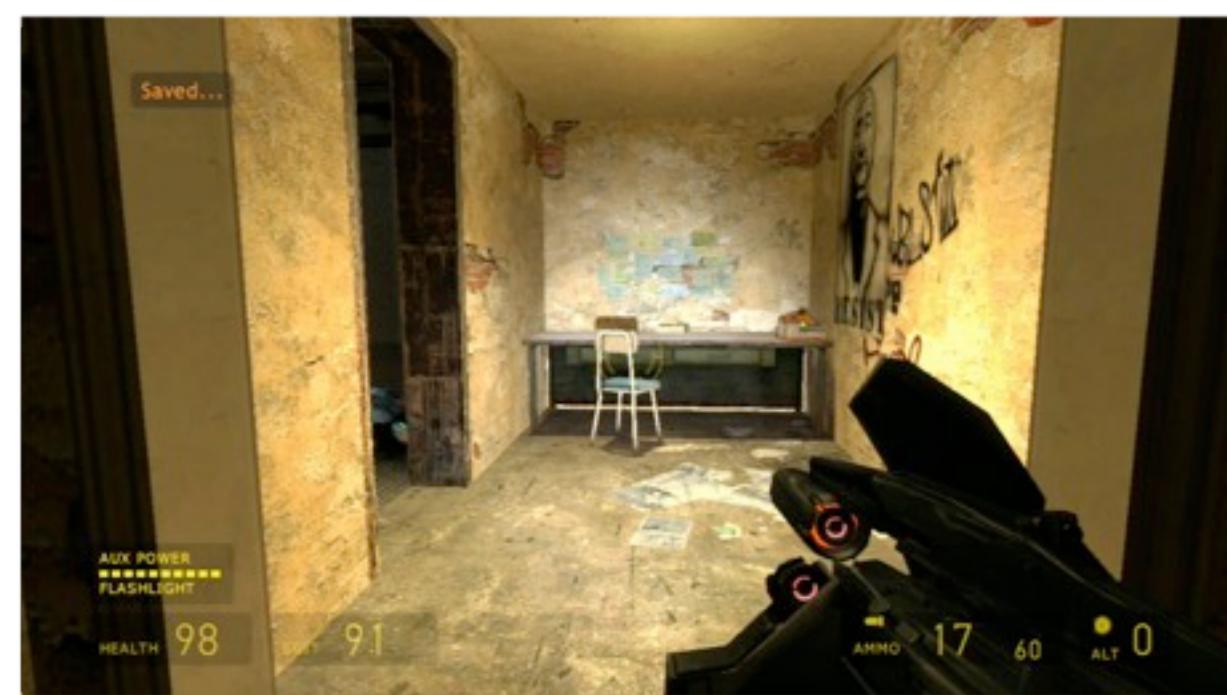
Article Discussion Read View source Search

Unwinnable

From Wikipedia, the free encyclopedia

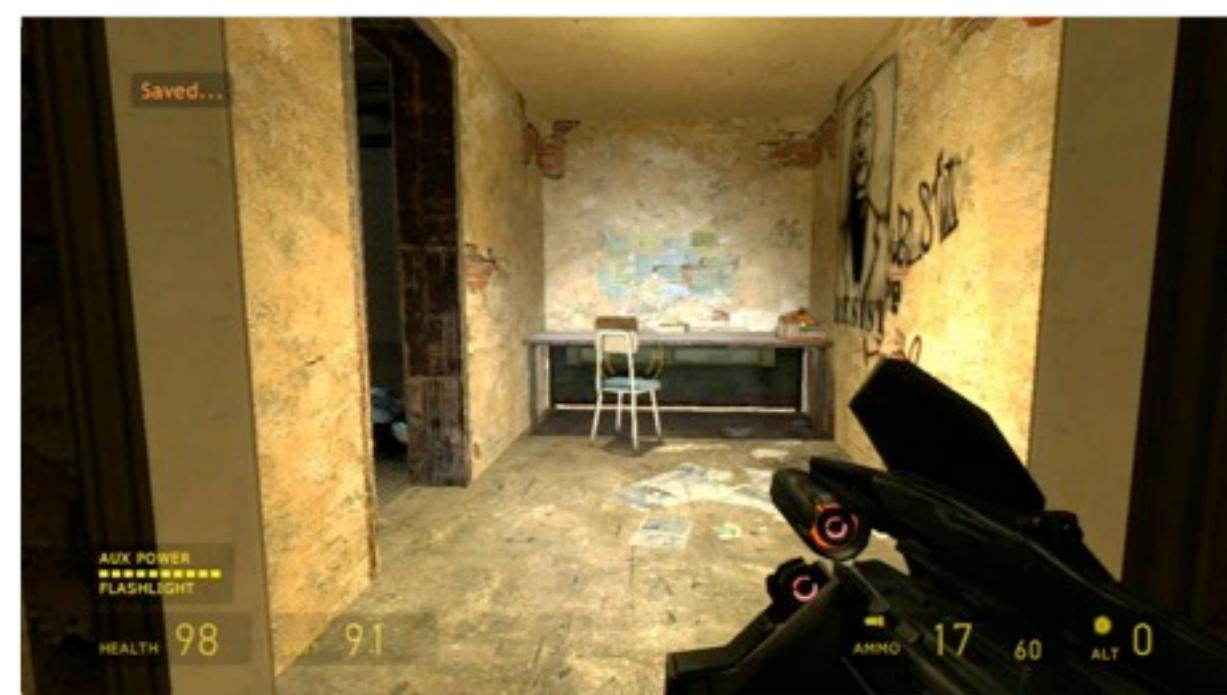
:(  
  
Unwinnable is a state in many [text adventures](#), [graphical adventure games](#) and [computer role-playing games](#) where it is impossible for the player to win the game (not due to a [bug](#) but by [design](#)), and where the only other options are restarting the game, loading a previously saved game, wandering indefinitely, or a [game over](#) (negative game end such as [death](#)). It is also known as a [walking dead](#), [dead end](#) or [zombie](#) situation.

# Video Game Analogy



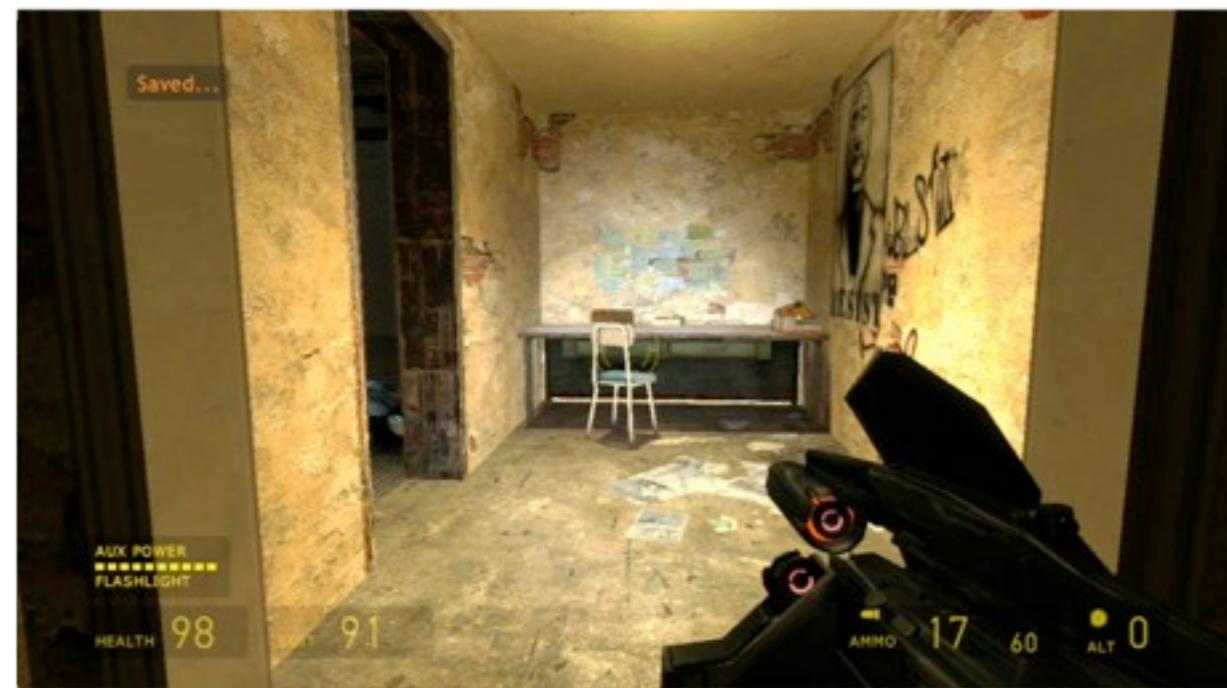
# Video Game Analogy

- Multiple, frequent saves:



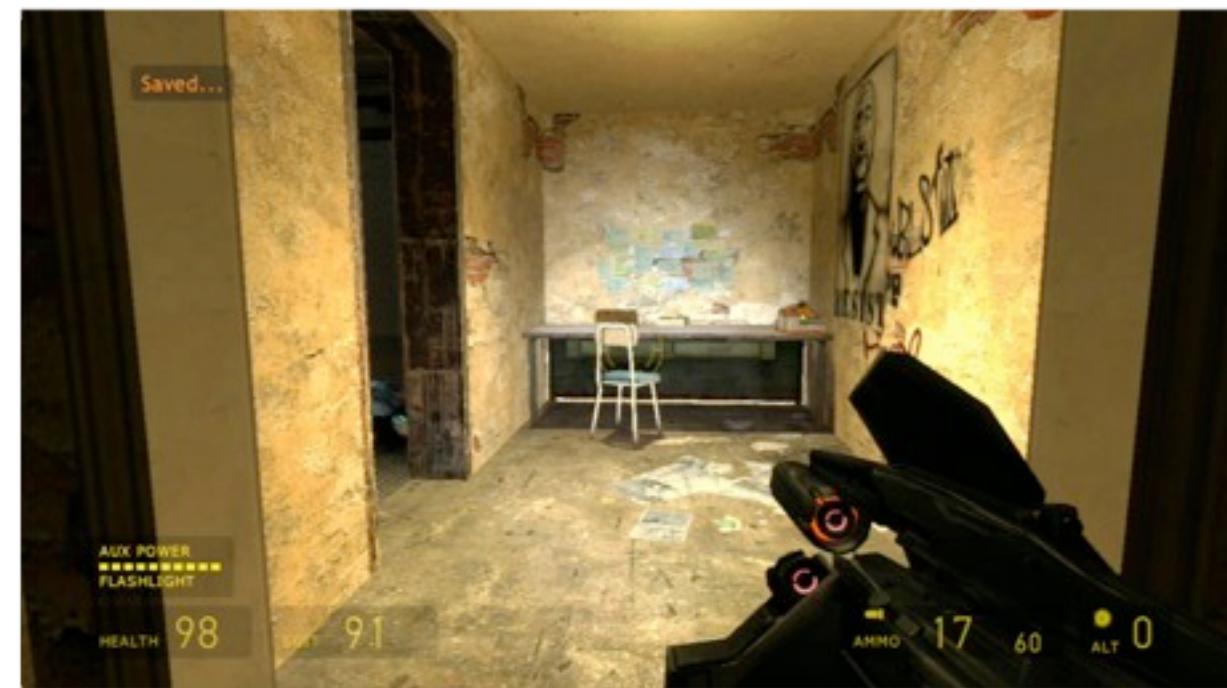
# Video Game Analogy

- Multiple, frequent saves:
  - Security to avoid lost progress



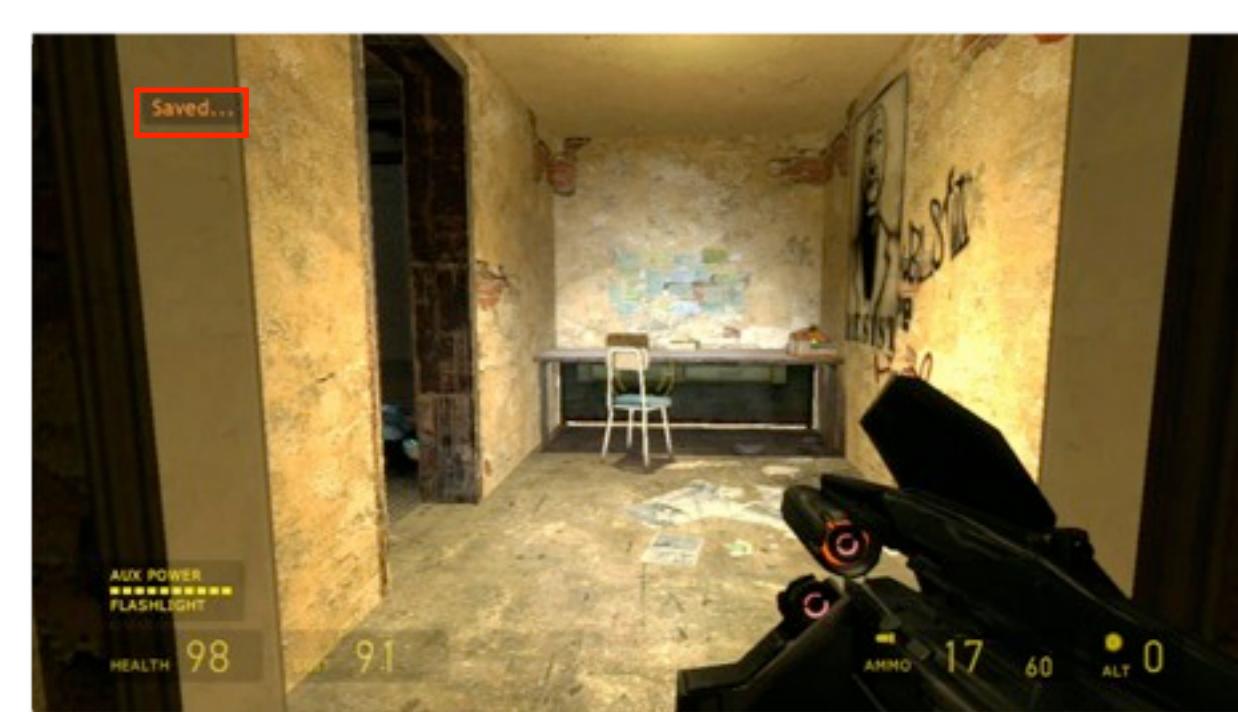
# Video Game Analogy

- Multiple, frequent saves:
  - Security to avoid lost progress
  - Ability to return to earlier state and choose to go down new path (branching)



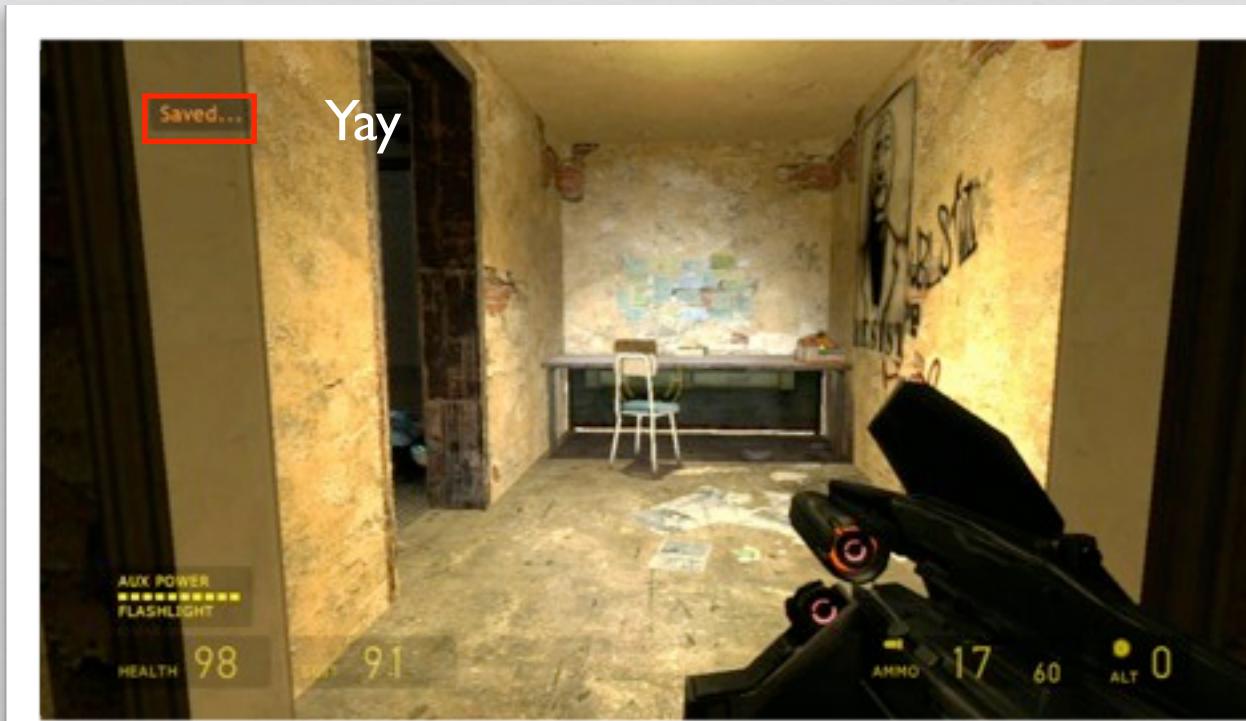
# Video Game Analogy

- Multiple, frequent saves:
  - Security to avoid lost progress
  - Ability to return to earlier state and choose to go down new path (branching)



# Video Game Analogy

- Multiple, frequent saves:
  - Security to avoid lost progress
  - Ability to return to earlier state and choose to go down new path (branching)



# Video Game Analogy

- Multiple, frequent saves:
  - Security to avoid lost progress
  - Ability to return to earlier state and choose to go down new path (branching)



This is the mentality we want:

Save early, save often



# **Why Version Control:**

# Why Version Control:

- Management of changes to programs, documents, and other computer files

# Why Version Control:

- Management of changes to programs, documents, and other computer files
- Checkpoints in evolution of source code

# Why Version Control:

- Management of changes to programs, documents, and other computer files
- Checkpoints in evolution of source code
- Backup, restore, synchronize, track changes, track ownership, branch, and merge

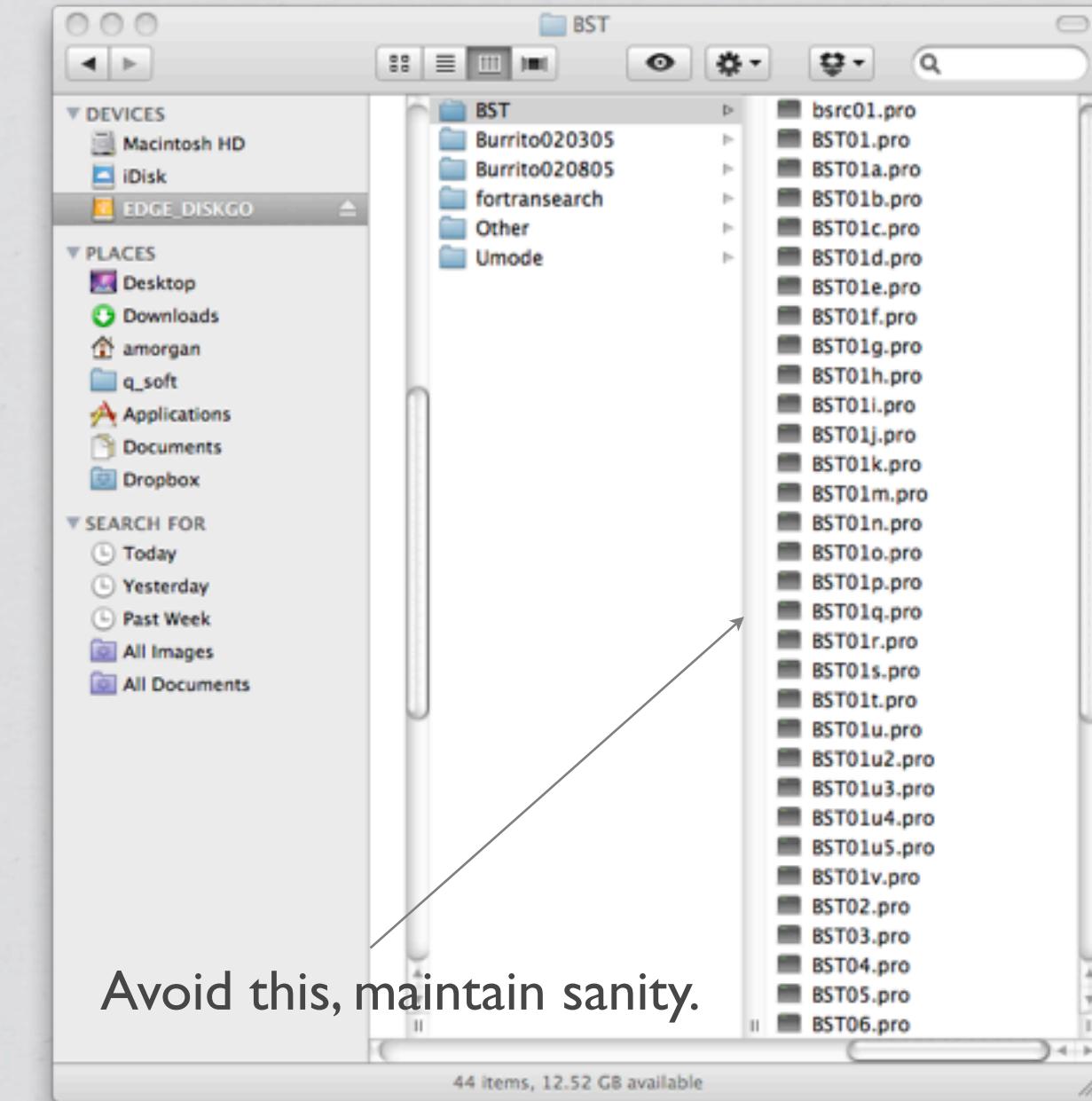
# Why Version Control:

- Management of changes to programs, documents, and other computer files
- Checkpoints in evolution of source code
- Backup, restore, synchronize, track changes, track ownership, branch, and merge
- Allow multiple developers to collaborate on a single codebase

# Why Version Control:

- Management of changes to programs, documents, and other computer files
- Checkpoints in evolution of source code
- Backup, restore, synchronize, track changes, track ownership, branch, and merge
- Allow multiple developers to collaborate on a single codebase
- Maintain sanity

# Primitive “Version Control”



Better than nothing, but far from perfect:

inconvenient (annoying),  
expensive (file space), and  
quickly spirals out of control.

# Common Terms - Setup

h/t <http://betterexplained.com/articles/a-visual-guide-to-version-control/>

# Common Terms - Setup

- **Repository (repo)**: Database storing all versions of all the files, and their changes

h/t <http://betterexplained.com/articles/a-visual-guide-to-version-control/>

# Common Terms - Setup

- **Repository (repo)**: Database storing all versions of all the files, and their changes
- **Server**: Computer storing the repository

h/t <http://betterexplained.com/articles/a-visual-guide-to-version-control/>

# Common Terms - Setup

- **Repository (repo)**: Database storing all versions of all the files, and their changes
- **Server**: Computer storing the repository
- **Client**: Computer connecting to repository

h/t <http://betterexplained.com/articles/a-visual-guide-to-version-control/>

# Common Terms - Setup

- **Repository (repo)**: Database storing all versions of all the files, and their changes
- **Server**: Computer storing the repository
- **Client**: Computer connecting to repository
- **Working Copy**: Local directory of files where you make changes

h/t <http://betterexplained.com/articles/a-visual-guide-to-version-control/>

# Common Terms - Setup

- **Repository (repo)**: Database storing all versions of all the files, and their changes
- **Server**: Computer storing the repository
- **Client**: Computer connecting to repository
- **Working Copy**: Local directory of files where you make changes
- **Trunk**: Primary location for the code in the repo

h/t <http://betterexplained.com/articles/a-visual-guide-to-version-control/>

# Common Terms - Basics

h/t <http://betterexplained.com/articles/a-visual-guide-to-version-control/>

# Common Terms - Basics

- **Add**: Put a file in the repo (begin tracking it)

h/t <http://betterexplained.com/articles/a-visual-guide-to-version-control/>

# Common Terms - Basics

- **Add**: Put a file in the repo (begin tracking it)
- **Revision**: Current version of a file (v1, v2, etc)

h/t <http://betterexplained.com/articles/a-visual-guide-to-version-control/>

# Common Terms - Basics

- **Add**: Put a file in the repo (begin tracking it)
- **Revision**: Current version of a file (v1, v2, etc)
- **Head**: Latest revision in the repository

h/t <http://betterexplained.com/articles/a-visual-guide-to-version-control/>

# Common Terms - Basics

- **Add**: Put a file in the repo (begin tracking it)
- **Revision**: Current version of a file (v1, v2, etc)
- **Head**: Latest revision in the repository
- **Check out**: Download a file (or files) from the repository

h/t <http://betterexplained.com/articles/a-visual-guide-to-version-control/>

# Common Terms - Basics

h/t <http://betterexplained.com/articles/a-visual-guide-to-version-control/>

# Common Terms - Basics

- **Check in/commit:** Upload a changed file to the repository
  - (it gets a new version number)

h/t <http://betterexplained.com/articles/a-visual-guide-to-version-control/>

# Common Terms - Basics

- **Check in/commit:** Upload a changed file to the repository
  - (it gets a new version number)
- **Update/Sync/Pull:** Synchronize your files with the latest from the repository

h/t <http://betterexplained.com/articles/a-visual-guide-to-version-control/>

# Common Terms - Basics

- **Check in/commit:** Upload a changed file to the repository
  - (it gets a new version number)
- **Update/Sync/Pull:** Synchronize your files with the latest from the repository
- **Revert:** Throw away local changes and reload latest repository version

h/t <http://betterexplained.com/articles/a-visual-guide-to-version-control/>

# Common Terms - Advanced

h/t <http://betterexplained.com/articles/a-visual-guide-to-version-control/>

# Common Terms - Advanced

- **Branch:** Create a separate copy of a file/folder for private use (bug fixing, testing, etc).

h/t <http://betterexplained.com/articles/a-visual-guide-to-version-control/>

# Common Terms - Advanced

- **Branch:** Create a separate copy of a file/folder for private use (bug fixing, testing, etc).
- **Diff:** Finding the differences between two files, for seeing what changed between revisions.

h/t <http://betterexplained.com/articles/a-visual-guide-to-version-control/>

# Common Terms - Advanced

- **Branch:** Create a separate copy of a file/folder for private use (bug fixing, testing, etc).
- **Diff:** Finding the differences between two files, for seeing what changed between revisions.
- **Merge/patch:** Apply the changes from one file to another, to bring it up-to-date.

h/t <http://betterexplained.com/articles/a-visual-guide-to-version-control/>

# Common Terms - Advanced

- **Branch:** Create a separate copy of a file/folder for private use (bug fixing, testing, etc).
- **Diff:** Finding the differences between two files, for seeing what changed between revisions.
- **Merge/patch:** Apply the changes from one file to another, to bring it up-to-date.
- **Conflict:** When changes contradict each other

h/t <http://betterexplained.com/articles/a-visual-guide-to-version-control/>

# Common Terms - Advanced

- **Branch:** Create a separate copy of a file/folder for private use (bug fixing, testing, etc).
- **Diff:** Finding the differences between two files, for seeing what changed between revisions.
- **Merge/patch:** Apply the changes from one file to another, to bring it up-to-date.
- **Conflict:** When changes contradict each other
- **Resolve:** Fixing the changes that contradict each other and committing the correct version

h/t <http://betterexplained.com/articles/a-visual-guide-to-version-control/>

# Git: Getting Started

Very easy to start version control of your files

- Initialize repository
- Add files
- Commit changes to repository

# Git: Getting Started

Very easy to start version control of your files

- Initialize repository
- Add files
- Commit changes to repository

```
$ cd mysoftware
```

# Git: Getting Started

Very easy to start version control of your files

- Initialize repository
- Add files
- Commit changes to repository

```
$ cd mysoftware  
$ git init
```

# Git: Getting Started

Very easy to start version control of your files

- Initialize repository
- Add files
- Commit changes to repository

```
$ cd mysoftware  
$ git init  
Initialized empty Git repository in [path]
```

# Git: Getting Started

Very easy to start version control of your files

- Initialize repository
- Add files
- Commit changes to repository

```
$ cd mysoftware  
$ git init  
Initialized empty Git repository in [path]  
$ ls -a
```

# Git: Getting Started

Very easy to start version control of your files

- Initialize repository
- Add files
- Commit changes to repository

```
$ cd mysoftware  
$ git init  
Initialized empty Git repository in [path]  
$ ls -a  
. .git [all your other files]
```

# Git: Getting Started

Very easy to start version control of your files

- Initialize repository
- Add files
- Commit changes to repository

```
$ cd mysoftware
$ git init
Initialized empty Git repository in [path]
$ ls -a
. .git [all your other files]
$ git add .
```

# Git: Getting Started

Very easy to start version control of your files

- Initialize repository
- Add files
- Commit changes to repository

```
$ cd mysoftware
$ git init
Initialized empty Git repository in [path]
$ ls -a
. .. .git [all your other files]
$ git add .
$ git commit -m "Initial import of my files"
```

# Git: Getting Started

Very easy to start version control of your files

- Initialize repository
- Add files
- Commit changes to repository

```
$ cd mysoftware
$ git init
Initialized empty Git repository in [path]
$ ls -a
. .git [all your other files]
$ git add .
$ git commit -m "Initial import of my files"
[master (root-commit) 2c7aded] Initial import of my files
```

# Git: Getting Started

Very easy to start version control of your files

- Initialize repository
- Add files
- Commit changes to repository

```
$ cd mysoftware
$ git init
Initialized empty Git repository in [path]
$ ls -a
. .git [all your other files]
$ git add .
$ git commit -m "Initial import of my files"
[master (root-commit) 2c7aded] Initial import of my files
Committer: Isaac Shivvers <ishivvers@berkeley.edu>
```

# Git: Basic Workflow

# Git: Basic Workflow

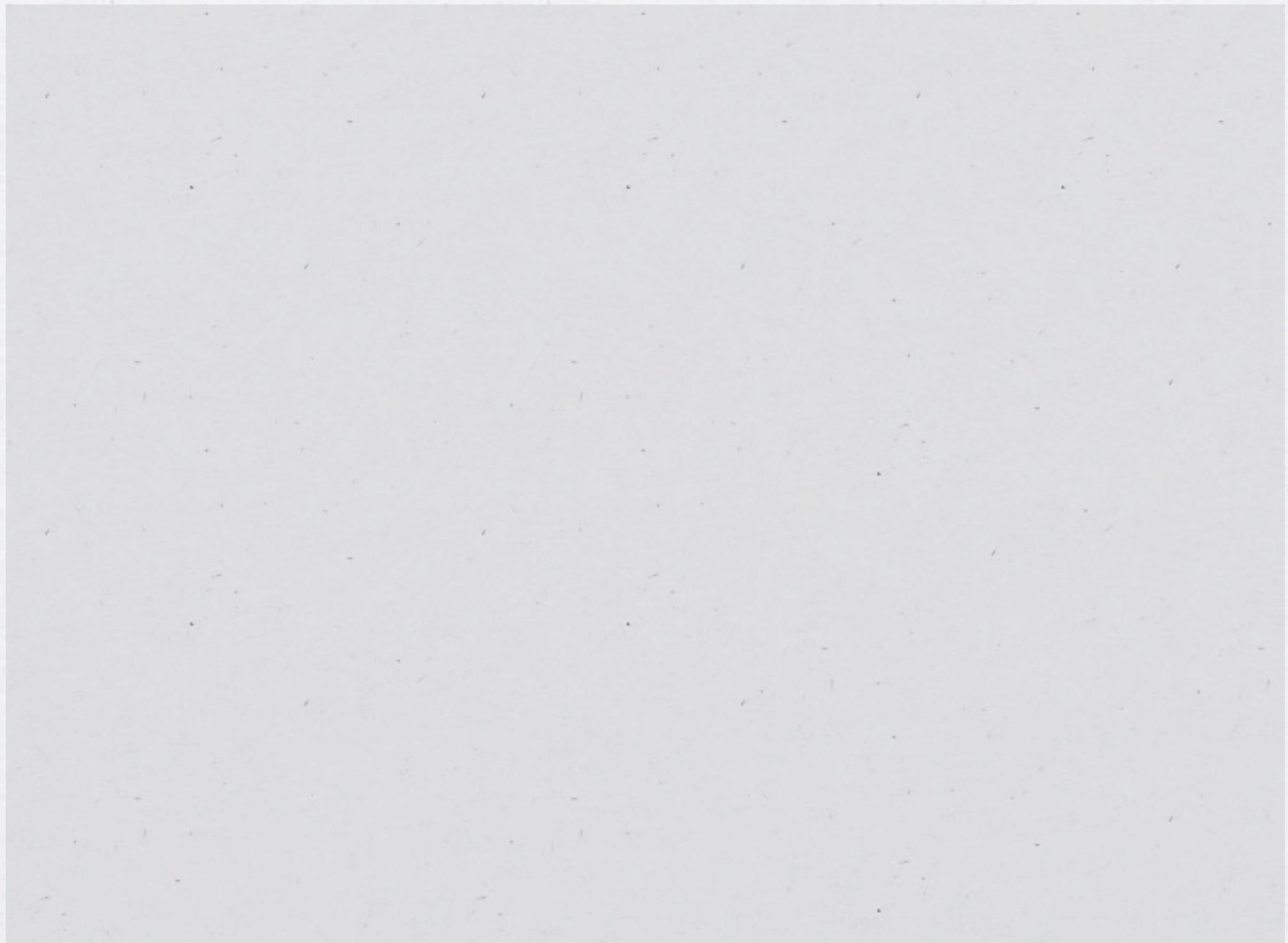
- Modify code/document

# Git: Basic Workflow

- Modify code/document
- Commit changes

# Git: Basic Workflow

- Modify code/document
- Commit changes
- Repeat



**Add**

# Add

```
$ echo "This is the readme" > README
```

# Add

```
$ echo "This is the readme" > README  
$ git add README
```

# Add

```
$ echo "This is the readme" > README
$ git add README
$ git commit -m "Added a readme file as a test"
```

# Add

```
$ echo "This is the readme" > README
$ git add README
$ git commit -m "Added a readme file as a test"
[master (root-commit) 2c7aded] Added a readme file as a test
```

# Add

```
$ echo "This is the readme" > README
$ git add README
$ git commit -m "Added a readme file as a test"
[master (root-commit) 2c7aded] Added a readme file as a test
 1 files changed, 1 insertions(+), 0 deletions(-)
```

# Add

```
$ echo "This is the readme" > README
$ git add README
$ git commit -m "Added a readme file as a test"
[master (root-commit) 2c7aded] Added a readme file as a test
 1 files changed, 1 insertions(+), 0 deletions(-)
 create mode 100644 README
```

# Add

```
$ echo "This is the readme" > README
$ git add README
$ git commit -m "Added a readme file as a test"
[master (root-commit) 2c7aded] Added a readme file as a test
 1 files changed, 1 insertions(+), 0 deletions(-)
 create mode 100644 README
```

# Delete

# Add

```
$ echo "This is the readme" > README
$ git add README
$ git commit -m "Added a readme file as a test"
[master (root-commit) 2c7aded] Added a readme file as a test
 1 files changed, 1 insertions(+), 0 deletions(-)
 create mode 100644 README
```

# Delete

```
{-- to just stop tracking changes --}
```

# Add

```
$ echo "This is the readme" > README
$ git add README
$ git commit -m "Added a readme file as a test"
[master (root-commit) 2c7aded] Added a readme file as a test
 1 files changed, 1 insertions(+), 0 deletions(-)
 create mode 100644 README
```

# Delete

```
{-- to just stop tracking changes --}
$ git rm uselessfile.txt
```

# Add

```
$ echo "This is the readme" > README
$ git add README
$ git commit -m "Added a readme file as a test"
[master (root-commit) 2c7aded] Added a readme file as a test
 1 files changed, 1 insertions(+), 0 deletions(-)
 create mode 100644 README
```

# Delete

```
{-- to just stop tracking changes --}
$ git rm uselessfile.txt
$ git rm -r abunchof/oldfiles/
```

# Add

```
$ echo "This is the readme" > README
$ git add README
$ git commit -m "Added a readme file as a test"
[master (root-commit) 2c7aded] Added a readme file as a test
 1 files changed, 1 insertions(+), 0 deletions(-)
 create mode 100644 README
```

# Delete

```
{-- to just stop tracking changes --}
$ git rm uselessfile.txt
$ git rm -r abunchof/oldfiles/

{-- to delete the file altogether --}
```

# Add

```
$ echo "This is the readme" > README
$ git add README
$ git commit -m "Added a readme file as a test"
[master (root-commit) 2c7aded] Added a readme file as a test
 1 files changed, 1 insertions(+), 0 deletions(-)
 create mode 100644 README
```

# Delete

```
{-- to just stop tracking changes --}
$ git rm uselessfile.txt
$ git rm -r abunchof/oldfiles/

{-- to delete the file altogether --}
$ git rm -f uselessfile.txt
```

# Add

```
$ echo "This is the readme" > README
$ git add README
$ git commit -m "Added a readme file as a test"
[master (root-commit) 2c7aded] Added a readme file as a test
 1 files changed, 1 insertions(+), 0 deletions(-)
 create mode 100644 README
```

# Delete

```
{-- to just stop tracking changes --}
$ git rm uselessfile.txt
$ git rm -r abunchof/oldfiles/

{-- to delete the file altogether --}
$ git rm -f uselessfile.txt
```

# Rename (same as delete + add)

# Add

```
$ echo "This is the readme" > README
$ git add README
$ git commit -m "Added a readme file as a test"
[master (root-commit) 2c7aded] Added a readme file as a test
 1 files changed, 1 insertions(+), 0 deletions(-)
 create mode 100644 README
```

# Delete

```
{-- to just stop tracking changes --}
$ git rm uselessfile.txt
$ git rm -r abunchof/oldfiles/

{-- to delete the file altogether --}
$ git rm -f uselessfile.txt
```

# Rename (same as delete + add)

```
$ git mv oldname.txt newname.txt
```

# Git Log

Get a list of all recent commits and their “SHA1 hashes”

```
commit 766f9881690d240ba334153047649b8b8f11c664
Author: Bob <bob@example.com>
Date: Tue Mar 14 01:59:26 2000 -0800
```

Replace printf() with write().

```
commit 82f5ea346a2e651544956a8653c0f58dc151275c
Author: Alice <alice@example.com>
Date: Thu Jan 1 00:00:00 1970 +0000
```

Initial commit.

- Unique 160-bit ID number for every string of bytes; useful for security.
- Each clone is a backup and corruptions will be spotted when attempting communication.

# Undoing Changes

# Undoing Changes

First few characters of hash enough to specify the commit

# Undoing Changes

First few characters of hash enough to specify the commit

- `git reset --hard`: (e.g. `git reset --hard 766f`) load an old commit and delete all commits newer than the one just loaded

# Undoing Changes

First few characters of hash enough to specify the commit

- **git reset --hard**: (e.g. git reset --hard 766f) load an old commit and delete all commits newer than the one just loaded
- **git checkout**: (e.g. git checkout 82f5) load an old commit, but new edits will be applied to this new branch and your other edits will still be accessible.

# Undoing Changes

First few characters of hash enough to specify the commit

- **git reset --hard**: (e.g. git reset --hard 766f) load an old commit and delete all commits newer than the one just loaded
- **git checkout**: (e.g. git checkout 82f5) load an old commit, but new edits will be applied to this new branch and your other edits will still be accessible.
- **git revert**: Throw away your local changes and reload the latest version from the repository.

# Undoing Changes

First few characters of hash enough to specify the commit

- **git reset --hard**: (e.g. git reset --hard 766f) load an old commit and delete all commits newer than the one just loaded
- **git checkout**: (e.g. git checkout 82f5) load an old commit, but new edits will be applied to this new branch and your other edits will still be accessible.
- **git revert**: Throw away your local changes and reload the latest version from the repository.
- Can refer to commits other ways; e.g. by start of commit message (git checkout :/"Replace printf()") or by, e.g., the 3rd last saved state (git checkout master~3)

# Git Status and Diff

```
$ git status
# On branch master
# Changed but not updated:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
#       modified: mystuff.py
#
no changes added to commit (use "git add" and/or "git commit -a")
```

```
$ git diff
diff --git a/mystuff.py b/mystuff.py
index 93ec7d1..7e9b565
--- a/mystuff.py
+++ b/mystuff.py
@@ -3,7 +3,10 @@
 print 'Hello, world!'

 for i in xrange (10):
-    print i, '...'
+    print 10 - i, '...'
+
+print '... blastoff!'
```

# Branching & Merging

**branch**: faster and more space efficient than cloning for situations when we need to switch gears and modify multiple versions of the code simultaneously

# Branching & Merging

**branch**: faster and more space efficient than cloning for situations when we need to switch gears and modify multiple versions of the code simultaneously

```
$ git status
```

# Branching & Merging

**branch**: faster and more space efficient than cloning for situations when we need to switch gears and modify multiple versions of the code simultaneously

```
$ git status  
# On branch master
```

# Branching & Merging

**branch**: faster and more space efficient than cloning for situations when we need to switch gears and modify multiple versions of the code simultaneously

```
$ git status  
# On branch master  
[ ... ]
```

# Branching & Merging

**branch**: faster and more space efficient than cloning for situations when we need to switch gears and modify multiple versions of the code simultaneously

```
$ git status  
# On branch master  
[...]  
  
$ git branch work-newalgo
```

# Branching & Merging

**branch**: faster and more space efficient than cloning for situations when we need to switch gears and modify multiple versions of the code simultaneously

```
$ git status  
# On branch master  
[...]  
  
$ git branch work-newalgo  
$ git checkout work-newalgo
```

# Branching & Merging

**branch**: faster and more space efficient than cloning for situations when we need to switch gears and modify multiple versions of the code simultaneously

```
$ git status  
# On branch master  
[...]  
  
$ git branch work-newalgo  
$ git checkout work-newalgo  
Switched to branch 'work-newalgo'
```

# Branching & Merging

**branch**: faster and more space efficient than cloning for situations when we need to switch gears and modify multiple versions of the code simultaneously

```
$ git status
# On branch master
[...]

$ git branch work-newalgo
$ git checkout work-newalgo
Switched to branch 'work-newalgo'

$ git checkout -b work-betterdocs
```

# Branching & Merging

**branch**: faster and more space efficient than cloning for situations when we need to switch gears and modify multiple versions of the code simultaneously

```
$ git status
# On branch master
[...]

$ git branch work-newalgo
$ git checkout work-newalgo
Switched to branch 'work-newalgo'

$ git checkout -b work-betterdocs
Switched to a new branch 'work-betterdocs'
```

# Branching & Merging

**branch**: faster and more space efficient than cloning for situations when we need to switch gears and modify multiple versions of the code simultaneously

```
$ git status
# On branch master
[...]

$ git branch work-newalgo
$ git checkout work-newalgo
Switched to branch 'work-newalgo'

$ git checkout -b work-betterdocs
Switched to a new branch 'work-betterdocs'
$ git branch
```

# Branching & Merging

**branch**: faster and more space efficient than cloning for situations when we need to switch gears and modify multiple versions of the code simultaneously

```
$ git status
# On branch master
[...]

$ git branch work-newalgo
$ git checkout work-newalgo
Switched to branch 'work-newalgo'

$ git checkout -b work-betterdocs
Switched to a new branch 'work-betterdocs'
$ git branch
  master
```

# Branching & Merging

**branch**: faster and more space efficient than cloning for situations when we need to switch gears and modify multiple versions of the code simultaneously

```
$ git status
# On branch master
[...]

$ git branch work-newalgo
$ git checkout work-newalgo
Switched to branch 'work-newalgo'

$ git checkout -b work-betterdocs
Switched to a new branch 'work-betterdocs'
$ git branch
  master
* work-betterdocs
```

# Branching & Merging

**branch**: faster and more space efficient than cloning for situations when we need to switch gears and modify multiple versions of the code simultaneously

```
$ git status
# On branch master
[...]

$ git branch work-newalgo
$ git checkout work-newalgo
Switched to branch 'work-newalgo'

$ git checkout -b work-betterdocs
Switched to a new branch 'work-betterdocs'
$ git branch
  master
* work-betterdocs
  work-newalgo
```

# Branching & Merging

**merge**: to combine branches back together

# Branching & Merging

**merge**: to combine branches back together

```
[ make a bunch of commits on work branch ]
```

# Branching & Merging

**merge**: to combine branches back together

```
[ make a bunch of commits on work branch ]  
$ git checkout master
```

# Branching & Merging

**merge**: to combine branches back together

```
[ make a bunch of commits on work branch ]  
  
$ git checkout master  
Switch to branch 'master'
```

# Branching & Merging

**merge**: to combine branches back together

```
[ make a bunch of commits on work branch ]  
  
$ git checkout master  
Switch to branch 'master'  
$ git merge work-betterdocs
```

# Branching & Merging

**merge**: to combine branches back together

```
[ make a bunch of commits on work branch ]  
  
$ git checkout master  
Switch to branch 'master'  
$ git merge work-betterdocs  
Updating 213a816..a9fae1e
```

# Branching & Merging

**merge**: to combine branches back together

```
[ make a bunch of commits on work branch ]  
  
$ git checkout master  
Switch to branch 'master'  
$ git merge work-betterdocs  
Updating 213a816..a9fae1e  
Fast-forward
```

# Branching & Merging

**merge**: to combine branches back together

```
[ make a bunch of commits on work branch ]  
  
$ git checkout master  
Switch to branch 'master'  
$ git merge work-betterdocs  
Updating 213a816..a9fae1e  
Fast-forward  
 README | 100 ++++++++
```

# Branching & Merging

**merge**: to combine branches back together

```
[ make a bunch of commits on work branch ]  
  
$ git checkout master  
Switch to branch 'master'  
$ git merge work-betterdocs  
Updating 213a816..a9fae1e  
Fast-forward  
 README | 100 ++++++++  
 LICENSE | 5+
```

# Branching & Merging

**merge**: to combine branches back together

```
[ make a bunch of commits on work branch ]  
  
$ git checkout master  
Switch to branch 'master'  
$ git merge work-betterdocs  
Updating 213a816..a9fae1e  
Fast-forward  
 README | 100 ++++++++  
 LICENSE | 5+  
 INSTALL | 120/30 ++++++-----
```

# Branching & Merging

**merge**: to combine branches back together

```
[ make a bunch of commits on work branch ]

$ git checkout master
Switch to branch 'master'
$ git merge work-betterdocs
Updating 213a816..a9fae1e
Fast-forward
 README |    100 ++++++++
 LICENSE |     5+
 INSTALL |   120/30 ++++++-----
 3 files changed, 225 insertions(+), 30 deletions(-)
```

# Branching & Merging

**merge**: to combine branches back together

```
[ make a bunch of commits on work branch ]  
  
$ git checkout master  
Switch to branch 'master'  
$ git merge work-betterdocs  
Updating 213a816..a9fae1e  
Fast-forward  
 README | 100 ++++++++  
 LICENSE | 5+  
 INSTALL | 120/30 +++++++-----  
 3 files changed, 225 insertions(+), 30 deletions(-)  
 create mode 100644 LICENSE
```

# Branching & Merging

**merge**: to combine branches back together

```
[ make a bunch of commits on work branch ]

$ git checkout master
Switch to branch 'master'
$ git merge work-betterdocs
Updating 213a816..a9fae1e
Fast-forward
 README |    100 ++++++++
 LICENSE |     5+
 INSTALL |   120/30 ++++++-----
 3 files changed, 225 insertions(+), 30 deletions(-)
 create mode 100644 LICENSE
 create mode 100644 README
```

# Branching & Merging

**merge**: to combine branches back together

```
[ make a bunch of commits on work branch ]

$ git checkout master
Switch to branch 'master'
$ git merge work-betterdocs
Updating 213a816..a9fae1e
Fast-forward
 README |    100 ++++++++
 LICENSE |     5+
 INSTALL |   120/30 ++++++-----
 3 files changed, 225 insertions(+), 30 deletions(-)
 create mode 100644 LICENSE
 create mode 100644 README
$ git branch -d work-betterdocs
```

# Branching & Merging

**merge**: to combine branches back together

```
[ make a bunch of commits on work branch ]

$ git checkout master
Switch to branch 'master'
$ git merge work-betterdocs
Updating 213a816..a9fae1e
Fast-forward
 README |    100 ++++++++
 LICENSE |     5+
 INSTALL |   120/30 ++++++-----
 3 files changed, 225 insertions(+), 30 deletions(-)
 create mode 100644 LICENSE
 create mode 100644 README
$ git branch -d work-betterdocs
Deleted branch work-betterdocs (was a9fae1e).
```

# Git GUI Tools

Unless you *really* like working from the command line, tools like `gitk` (a history visualizer) and `git gui` (a GUI interface to git's functionality) are incredibly helpful.

Those two are included with git, but there are many other options as well:

[https://git.wiki.kernel.org/index.php/  
Interfaces,\\_frontends,\\_and\\_tools#Graphical\\_Interfaces](https://git.wiki.kernel.org/index.php/Interfaces,_frontends,_and_tools#Graphical_Interfaces)

# Git GUI Tools

Unless you *really* like working from the command line, tools like `gitk` (a history visualizer) and `git gui` (a GUI interface to git's functionality) are incredibly helpful.

```
{-- from inside of working directory --}
```

Those two are included with git, but there are many other options as well:

[https://git.wiki.kernel.org/index.php/  
Interfaces,\\_frontends,\\_and\\_tools#Graphical\\_Interfaces](https://git.wiki.kernel.org/index.php/Interfaces,_frontends,_and_tools#Graphical_Interfaces)

# Git GUI Tools

Unless you *really* like working from the command line, tools like `gitk` (a history visualizer) and `git gui` (a GUI interface to git's functionality) are incredibly helpful.

```
{-- from inside of working directory --}  
$ gitk &
```

Those two are included with git, but there are many other options as well:

[https://git.wiki.kernel.org/index.php/  
Interfaces,\\_frontends,\\_and\\_tools#Graphical\\_Interfaces](https://git.wiki.kernel.org/index.php/Interfaces,_frontends,_and_tools#Graphical_Interfaces)

# Git GUI Tools

Unless you *really* like working from the command line, tools like `gitk` (a history visualizer) and `git gui` (a GUI interface to git's functionality) are incredibly helpful.

```
{-- from inside of working directory --}
$ gitk &

{-- and --}
```

Those two are included with git, but there are many other options as well:

[https://git.wiki.kernel.org/index.php/  
Interfaces,\\_frontends,\\_and\\_tools#Graphical\\_Interfaces](https://git.wiki.kernel.org/index.php/Interfaces,_frontends,_and_tools#Graphical_Interfaces)

# Git GUI Tools

Unless you *really* like working from the command line, tools like `gitk` (a history visualizer) and `git gui` (a GUI interface to git's functionality) are incredibly helpful.

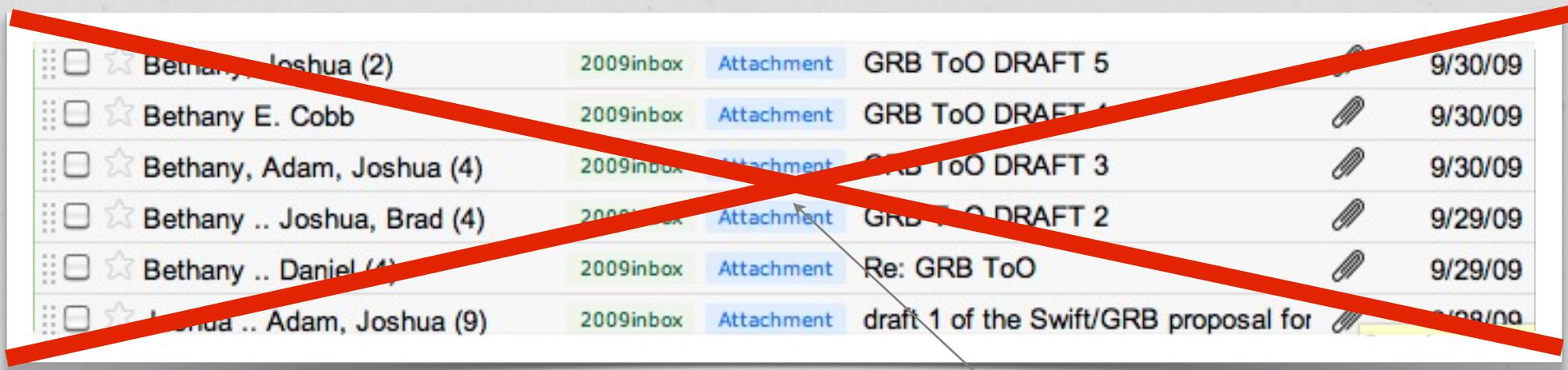
```
{-- from inside of working directory --}
$ gitk &

{-- and --}
$ git gui &
```

Those two are included with git, but there are many other options as well:

[https://git.wiki.kernel.org/index.php/  
Interfaces,\\_frontends,\\_and\\_tools#Graphical\\_Interfaces](https://git.wiki.kernel.org/index.php/Interfaces,_frontends,_and_tools#Graphical_Interfaces)

# Collaboration



<input type="checkbox"/> ★ Bethany, Joshua (2)	2009inbox	Attachment	GRB ToO DRAFT 5		9/30/09
<input type="checkbox"/> ★ Bethany E. Cobb	2009inbox	Attachment	GRB ToO DRAFT 4		9/30/09
<input type="checkbox"/> ★ Bethany, Adam, Joshua (4)	2009inbox	Attachment	GRB ToO DRAFT 3		9/30/09
<input type="checkbox"/> ★ Bethany .. Joshua, Brad (4)	2009inbox	Attachment	GRB ToO DRAFT 2		9/29/09
<input type="checkbox"/> ★ Bethany .. Daniel (4)	2009inbox	Attachment	Re: GRB ToO		9/29/09
<input type="checkbox"/> ★ Bethany .. Adam, Joshua (9)	2009inbox	Attachment	draft 1 of the Swift/GRB proposal for		9/29/09

Avoid this, maintain sanity.

# Collaboration

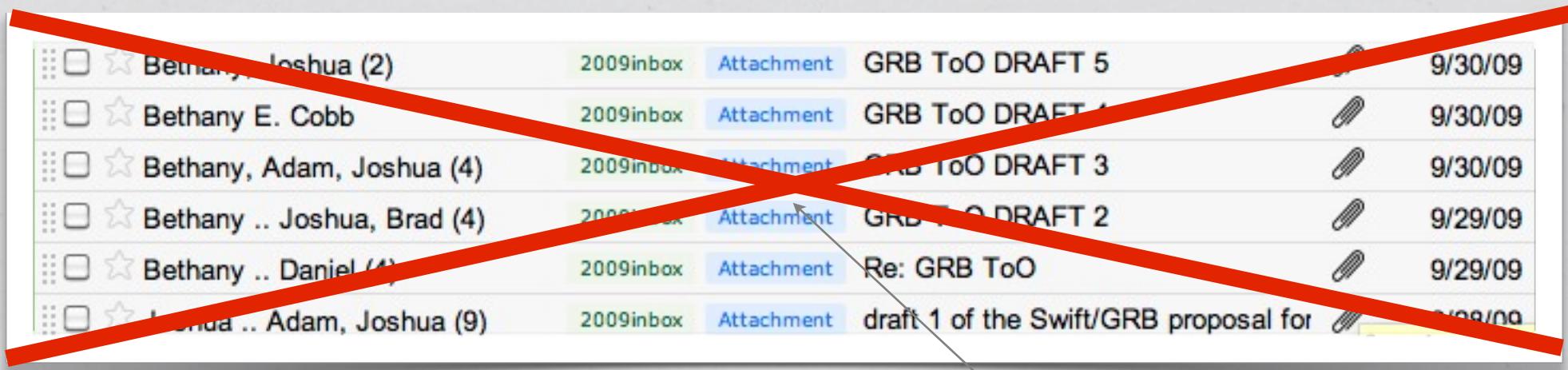
- Git makes it easy to work on a single code/document with multiple people contributing (or similarly, one person developing the same code on multiple computers)

✉ ★ Bethany, Joshua (2)	2009inbox	Attachment	GRB ToO DRAFT 5	9/30/09
✉ ★ Bethany E. Cobb	2009inbox	Attachment	GRB ToO DRAFT 4	9/30/09
✉ ★ Bethany, Adam, Joshua (4)	2009inbox	Attachment	GRB ToO DRAFT 3	9/30/09
✉ ★ Bethany .. Joshua, Brad (4)	2009inbox	Attachment	GRB ToO DRAFT 2	9/29/09
✉ ★ Bethany .. Daniel (4)	2009inbox	Attachment	Re: GRB ToO	9/29/09
✉ ★ Joshua .. Adam, Joshua (9)	2009inbox	Attachment	draft 1 of the Swift/GRB proposal for	9/29/09

Avoid this, maintain sanity.

# Collaboration

- Git makes it easy to work on a single code/document with multiple people contributing (or similarly, one person developing the same code on multiple computers)
- Easily see who changed what and when, merge changes behind the scenes, and deal with conflicts when they arise



✉ ★ Bethany, Joshua (2)	2009inbox	Attachment	GRB ToO DRAFT 5	9/30/09
✉ ★ Bethany E. Cobb	2009inbox	Attachment	GRB ToO DRAFT 4	9/30/09
✉ ★ Bethany, Adam, Joshua (4)	2009inbox	Attachment	GRB ToO DRAFT 3	9/30/09
✉ ★ Bethany .. Joshua, Brad (4)	2009inbox	Attachment	GRB ToO DRAFT 2	9/29/09
✉ ★ Bethany .. Daniel (4)	2009inbox	Attachment	Re: GRB ToO	9/29/09
✉ ★ Joshua .. Adam, Joshua (9)	2009inbox	Attachment	draft 1 of the Swift/GRB proposal for	9/29/09

Avoid this, maintain sanity.

# Collaborating: Clones

- In distributed VCS, **cloning** is the standard operation to get files. You mirror the entire project in the new folder.
- Cloning useful for backups (even when not collaborating)

h/t “Git Magic”

# Collaborating: Clones

- In distributed VCS, **cloning** is the standard operation to get files. You mirror the entire project in the new folder.
- Cloning useful for backups (even when not collaborating)

```
# To clone a repository from other.computer:
```

h/t “Git Magic”

# Collaborating: Clones

- In distributed VCS, **cloning** is the standard operation to get files. You mirror the entire project in the new folder.
- Cloning useful for backups (even when not collaborating)

```
# To clone a repository from other.computer:  
$ git clone other.computer:/path/to/files
```

h/t “Git Magic”

# Collaborating: Clones

- In distributed VCS, **cloning** is the standard operation to get files. You mirror the entire project in the new folder.
- Cloning useful for backups (even when not collaborating)

```
# To clone a repository from other.computer:  
$ git clone other.computer:/path/to/files  
# From now on, to pull the state of files from another computer to
```

h/t “Git Magic”

# Collaborating: Clones

- In distributed VCS, **cloning** is the standard operation to get files. You mirror the entire project in the new folder.
- Cloning useful for backups (even when not collaborating)

```
# To clone a repository from other.computer:  
$ git clone other.computer:/path/to/files  
# From now on, to pull the state of files from another computer to  
# the one you're working on, do:
```

h/t “Git Magic”

# Collaborating: Clones

- In distributed VCS, **cloning** is the standard operation to get files. You mirror the entire project in the new folder.
- Cloning useful for backups (even when not collaborating)

```
# To clone a repository from other.computer:  
$ git clone other.computer:/path/to/files  
# From now on, to pull the state of files from another computer to  
# the one you're working on, do:  
$ git commit -a
```

h/t “Git Magic”

# Collaborating: Clones

- In distributed VCS, **cloning** is the standard operation to get files. You mirror the entire project in the new folder.
- Cloning useful for backups (even when not collaborating)

```
# To clone a repository from other.computer:  
$ git clone other.computer:/path/to/files  
# From now on, to pull the state of files from another computer to  
# the one you're working on, do:  
$ git commit -a  
$ git pull other.computer:/path/to/files HEAD
```

h/t “Git Magic”

# Collaborating: Clones

- In distributed VCS, **cloning** is the standard operation to get files. You mirror the entire project in the new folder.
- Cloning useful for backups (even when not collaborating)

```
# To clone a repository from other.computer:  
$ git clone other.computer:/path/to/files  
# From now on, to pull the state of files from another computer to  
# the one you're working on, do:  
$ git commit -a  
$ git pull other.computer:/path/to/files HEAD  
# where HEAD is a cursor that points to the latest commit
```

h/t “Git Magic”

# Collaborating: Hosting Projects

# Collaborating: Hosting Projects

- Because Git is distributed system, in order to be able to **push** your git projects back so that others can see your changes, you must have a host repo somewhere.

# Collaborating: Hosting Projects

- Because Git is distributed system, in order to be able to **push** your git projects back so that others can see your changes, you must have a host repo somewhere.
- **push** only to remote branches that are not currently checked out on the other side to avoid confusion. Could be safe and just push to **bare repositories** which have no working copy.

# Collaborating: Hosting Projects

- Because Git is distributed system, in order to be able to **push** your git projects back so that others can see your changes, you must have a host repo somewhere.
- **push** only to remote branches that are not currently checked out on the other side to avoid confusion. Could be safe and just push to **bare repositories** which have no working copy.
- Can either have a central server where everyone has ssh access, or one of the many free hosting sites if your code is public/open source (see <https://git.wiki.kernel.org/index.php/GitHosting>)

# Hosting Projects: GitHub

# Hosting Projects: GitHub

- For this class, we'll use GitHub as a standard, so let's go through a complete example

# Hosting Projects: GitHub

- For this class, we'll use GitHub as a standard, so let's go through a complete example
- log into GitHub, create a new repository (w/o a readme yet)

# Hosting Projects: GitHub

- For this class, we'll use GitHub as a standard, so let's go through a complete example
- log into GitHub, create a new repository (w/o a readme yet)
- on the command line, create some files, and push to this new repository

# Hosting Projects: GitHub

- For this class, we'll use GitHub as a standard, so let's go through a complete example
- log into GitHub, create a new repository (w/o a readme yet)
- on the command line, create some files, and push to this new repository

```
$ nano README
```

# Hosting Projects: GitHub

- For this class, we'll use GitHub as a standard, so let's go through a complete example
- log into GitHub, create a new repository (w/o a readme yet)
- on the command line, create some files, and push to this new repository

```
$ nano README  
$ git init
```

# Hosting Projects: GitHub

- For this class, we'll use GitHub as a standard, so let's go through a complete example
- log into GitHub, create a new repository (w/o a readme yet)
- on the command line, create some files, and push to this new repository

```
$ nano README  
$ git init  
$ git add README
```

# Hosting Projects: GitHub

- For this class, we'll use GitHub as a standard, so let's go through a complete example
- log into GitHub, create a new repository (w/o a readme yet)
- on the command line, create some files, and push to this new repository

```
$ nano README  
$ git init  
$ git add README  
$ git commit -m "first commit"
```

# Hosting Projects: GitHub

- For this class, we'll use GitHub as a standard, so let's go through a complete example
- log into GitHub, create a new repository (w/o a readme yet)
- on the command line, create some files, and push to this new repository

```
$ nano README
$ git init
$ git add README
$ git commit -m "first commit"
$ git remote add origin https://github.com/[user_name]/[repo_name].git
```

# Hosting Projects: GitHub

- For this class, we'll use GitHub as a standard, so let's go through a complete example
- log into GitHub, create a new repository (w/o a readme yet)
- on the command line, create some files, and push to this new repository

```
$ nano README
$ git init
$ git add README
$ git commit -m "first commit"
$ git remote add origin https://github.com/[user_name]/[repo_name].git
$ git push origin master
```

# Updating Code

- After the bare repository is hosted on another server, **clone** a copy to your local disk
- **Pull** any changes from other users
- **Push** any new changes you make to the new server

# Updating Code

- After the bare repository is hosted on another server, **clone** a copy to your local disk
- **Pull** any changes from other users
- **Push** any new changes you make to the new server

```
$ cd myclone
```

# Updating Code

- After the bare repository is hosted on another server, **clone** a copy to your local disk
- **Pull** any changes from other users
- **Push** any new changes you make to the new server

```
$ cd myclone  
$ git clone https://github.com/ishivvers/test_this.git
```

# Updating Code

- After the bare repository is hosted on another server, **clone** a copy to your local disk
- **Pull** any changes from other users
- **Push** any new changes you make to the new server

```
$ cd myclone  
$ git clone https://github.com/ishivvers/test_this.git  
Cloning into test_this...
```

# Updating Code

- After the bare repository is hosted on another server, **clone** a copy to your local disk
- **Pull** any changes from other users
- **Push** any new changes you make to the new server

```
$ cd myclone
$ git clone https://github.com/ishivvers/test_this.git
Cloning into test_this...
# Make some changes....
```

# Updating Code

- After the bare repository is hosted on another server, **clone** a copy to your local disk
- **Pull** any changes from other users
- **Push** any new changes you make to the new server

```
$ cd myclone
$ git clone https://github.com/ishivvers/test_this.git
Cloning into test_this...
# Make some changes....
$ git commit -m "Long and descriptive commit message"
```

# Updating Code

- After the bare repository is hosted on another server, **clone** a copy to your local disk
- **Pull** any changes from other users
- **Push** any new changes you make to the new server

```
$ cd myclone
$ git clone https://github.com/ishivvers/test_this.git
Cloning into test_this...
# Make some changes....
$ git commit -m "Long and descriptive commit message"
# Pull to update to latest version of the code
```

# Updating Code

- After the bare repository is hosted on another server, **clone** a copy to your local disk
- **Pull** any changes from other users
- **Push** any new changes you make to the new server

```
$ cd myclone
$ git clone https://github.com/ishivvers/test_this.git
Cloning into test_this...
# Make some changes....
$ git commit -m "Long and descriptive commit message"
# Pull to update to latest version of the code
$ git pull origin master
```

# Updating Code

- After the bare repository is hosted on another server, **clone** a copy to your local disk
- **Pull** any changes from other users
- **Push** any new changes you make to the new server

```
$ cd myclone
$ git clone https://github.com/ishivvers/test_this.git
Cloning into test_this...
# Make some changes....
$ git commit -m "Long and descriptive commit message"
# Pull to update to latest version of the code
$ git pull origin master
# Resolve any merge conflicts and then commit them, if necessary
```

# Updating Code

- After the bare repository is hosted on another server, **clone** a copy to your local disk
- **Pull** any changes from other users
- **Push** any new changes you make to the new server

```
$ cd myclone
$ git clone https://github.com/ishivvers/test_this.git
Cloning into test_this...
# Make some changes....
$ git commit -m "Long and descriptive commit message"
# Pull to update to latest version of the code
$ git pull origin master
# Resolve any merge conflicts and then commit them, if necessary
# Push to check-in local changes to the hosted repository:
$ git push origin master
```

# Conflicts

- Generally, git will happily deal with multiple workers on the same file and merge the changes automatically when you [pull](#) from the central repository
- But when the same line is edited by multiple people, human intervention is generally needed to resolve

# Conflicts

- Generally, git will happily deal with multiple workers on the same file and merge the changes automatically when you [pull](#) from the central repository
- But when the same line is edited by multiple people, human intervention is generally needed to resolve

```
person1 = 'Alice'
```

# Conflicts

- Generally, git will happily deal with multiple workers on the same file and merge the changes automatically when you [pull](#) from the central repository
- But when the same line is edited by multiple people, human intervention is generally needed to resolve

```
person1 = 'Alice'  
person2 = 'Bob'
```

# Conflicts

- Generally, git will happily deal with multiple workers on the same file and merge the changes automatically when you [pull](#) from the central repository
- But when the same line is edited by multiple people, human intervention is generally needed to resolve

```
person1 = 'Alice'  
person2 = 'Bob'  
  
<<<<< alice:example.py
```

# Conflicts

- Generally, git will happily deal with multiple workers on the same file and merge the changes automatically when you [pull](#) from the central repository
- But when the same line is edited by multiple people, human intervention is generally needed to resolve

```
person1 = 'Alice'  
person2 = 'Bob'  
  
<<<<< alice:example.py  
print "Clearly the best person is: " + person1
```

# Conflicts

- Generally, git will happily deal with multiple workers on the same file and merge the changes automatically when you [pull](#) from the central repository
- But when the same line is edited by multiple people, human intervention is generally needed to resolve

```
person1 = 'Alice'  
person2 = 'Bob'  
  
<<<<< alice:example.py  
print "Clearly the best person is: " + person1  
=====
```

# Conflicts

- Generally, git will happily deal with multiple workers on the same file and merge the changes automatically when you [pull](#) from the central repository
- But when the same line is edited by multiple people, human intervention is generally needed to resolve

```
person1 = 'Alice'  
person2 = 'Bob'  
  
<<<<< alice:example.py  
print "Clearly the best person is: " + person1  
=====  
print "Clearly the best person is: " + person2
```

# Conflicts

- Generally, git will happily deal with multiple workers on the same file and merge the changes automatically when you [pull](#) from the central repository
- But when the same line is edited by multiple people, human intervention is generally needed to resolve

```
person1 = 'Alice'  
person2 = 'Bob'  
  
<<<<< alice:example.py  
print "Clearly the best person is: " + person1  
=====  
print "Clearly the best person is: " + person2  
>>>>> bob:example.txt
```

# Git: Collaboration Workflow

# Git: Collaboration Workflow

- Pull latest version from central repo

# Git: Collaboration Workflow

- Pull latest version from central repo
- Modify code/document

# Git: Collaboration Workflow

- Pull latest version from central repo
- Modify code/document
- Commit changes (early and often!)

# Git: Collaboration Workflow

- Pull latest version from central repo
- Modify code/document
- Commit changes (early and often!)
- Pull (again, and resolve any merge conflicts)

# Git: Collaboration Workflow

- Pull latest version from central repo
- Modify code/document
- Commit changes (early and often!)
- Pull (again, and resolve any merge conflicts)
- Push your changes to the central repo

# Git: Collaboration Workflow

- Pull latest version from central repo
- Modify code/document
- Commit changes (early and often!)
- Pull (again, and resolve any merge conflicts)
- Push your changes to the central repo
- Repeat

# Git ninja mastery

## getting help

```
$ git help [subcommand]
```

## browsing history

```
$ git log [-S]  
$ git show [commit]  
$ git blame [file]
```

## changing history

```
$ git commit --amend  
$ git reset  
$ git rebase [-i]
```

## working with other vcs tools

```
$ git cvsimport  
$ git svn
```

## advanced collaboration

```
$ git cherry-pick  
$ git format-patch  
$ git rerere  
$ git mergetool  
$ git send-email  
$ git stash
```

## tagging

```
$ git tag v1.2  
$ git push --tags
```

## shell script tools

```
$ git ls-files  
$ git describe  
$ git grep  
$ git rev-list
```

h/t Peter Williams

# In conclusion:

# In conclusion:

- Version Control is awesome and everyone should do it

# In conclusion:

- Version Control is awesome and everyone should do it
- Git is a free, actively developed distributed VCS

# In conclusion:

- Version Control is awesome and everyone should do it
- Git is a free, actively developed distributed VCS
- GUI tools make it easier

# In conclusion:

- Version Control is awesome and everyone should do it
- Git is a free, actively developed distributed VCS
- GUI tools make it easier
- Online repositories like GitHub allow for free backup of your codebase and easy collaboration

# In conclusion:

- Version Control is awesome and everyone should do it
- Git is a free, actively developed distributed VCS
- GUI tools make it easier
- Online repositories like GitHub allow for free backup of your codebase and easy collaboration
- Pull, Modify, Commit, Pull, Push, Repeat

# In conclusion:

- Version Control is awesome and everyone should do it
- Git is a free, actively developed distributed VCS
- GUI tools make it easier
- Online repositories like GitHub allow for free backup of your codebase and easy collaboration
- Pull, Modify, Commit, Pull, Push, Repeat

Git: Obtain the primary software (mac, linux, windows): <http://git-scm.com/>

Git Magic: a great online book introducing the software <http://www-cs-students.stanford.edu/~blynn/gitmagic/>

A list of graphical front-end clients for Git: [https://git.wiki.kernel.org/index.php/Interfaces,\\_frontends,\\_and\\_tools#Graphical\\_Interfaces](https://git.wiki.kernel.org/index.php/Interfaces,_frontends,_and_tools#Graphical_Interfaces)

Github: Online hosting service for Git repositories. <https://github.com/>

# Outline

- Managing Packages -  
`pip, setup.py, virtualenv`
  - Command Line Parsing - `argparse`
  - Building Modules & Packages  
  
`<breakout session>`
  - Version Control - `git`
- Debugging & Testing - `pdb, ipy %debug, pylint, pep8`
- Distribution - `distutils2`

# Debugging Code



<http://docs.python.org/library/pdb.html>

# Debugging Code

- Standard technique of inserting “print” statements may be inconvenient if code takes a long time to run



<http://docs.python.org/library/pdb.html>

# Debugging Code

- Standard technique of inserting “print” statements may be inconvenient if code takes a long time to run
- The module `pdb` is an interactive source debugger



<http://docs.python.org/library/pdb.html>

# Debugging Code

- Standard technique of inserting “print” statements may be inconvenient if code takes a long time to run
- The module `pdb` is an interactive source debugger
- Variables are preserved at breakpoint, and can interactively step through lines of code



<http://docs.python.org/library/pdb.html>

# Debugging Code

Use `pdb.set_trace()` to explore and interact with problematic code

<http://docs.python.org/library/pdb.html>

# Debugging Code

Use `pdb.set_trace()` to explore and interact with problematic code

```
[ in some problematic python file ]
```

<http://docs.python.org/library/pdb.html>

# Debugging Code

Use `pdb.set_trace()` to explore and interact with problematic code

```
[ in some problematic python file ]  
import pdb
```

<http://docs.python.org/library/pdb.html>

# Debugging Code

Use `pdb.set_trace()` to explore and interact with problematic code

```
[ in some problematic python file ]  
  
import pdb  
  
[ code code code ]
```

<http://docs.python.org/library/pdb.html>

# Debugging Code

Use `pdb.set_trace()` to explore and interact with problematic code

```
[ in some problematic python file ]  
  
import pdb  
  
[ code code code ]  
  
pdb.set_trace()
```

<http://docs.python.org/library/pdb.html>

# Debugging Code

Use `pdb.set_trace()` to explore and interact with problematic code

```
[ in some problematic python file ]  
  
import pdb  
  
[ code code code ]  
  
pdb.set_trace()  
  
[ problematic code ]
```

<http://docs.python.org/library/pdb.html>

# Debugging Code



# Debugging Code

- Can debug within iPython by typing `debug` after exception is raised



# Debugging Code

- Can debug within iPython by typing `debug` after exception is raised
- ‘help’ shows the commands available, for both pdb and iPython’s ipdb:



# Debugging Code

- Can debug within iPython by typing `debug` after exception is raised
- ‘help’ shows the commands available, for both pdb and iPython’s ipdb:

```
ipdb> help

Documented commands (type help <topic>):
=====
EOF      bt          cont      enable    jump     pdef      r        tbreak   w
a         c          continue  exit      l        pdoc      restart  u        whatis
alias    cl          d          h         list     pinfo     return  unalias where
args    clear      debug      help      n        pp       run      unt
b       commands  disable   ignore   next     q        s       until
break  condition down      j        p       quit     step     up
```



# Documentation

<http://docs.python.org/distutils/index.html>

# Documentation

- Good documentation is important if anyone else needs to use your code, and when the inevitable time comes that you forget what it does as well

<http://docs.python.org/distutils/index.html>

# Documentation

- Good documentation is important if anyone else needs to use your code, and when the inevitable time comes that you forget what it does as well
- Docstrings make it easy to insert documentation right into your code

<http://docs.python.org/distutils/index.html>

# Documentation

- Good documentation is important if anyone else needs to use your code, and when the inevitable time comes that you forget what it does as well
- Docstrings make it easy to insert documentation right into your code

```
def important_function(a=0.0, b=0.0):
```

<http://docs.python.org/distutils/index.html>

# Documentation

- Good documentation is important if anyone else needs to use your code, and when the inevitable time comes that you forget what it does as well
- Docstrings make it easy to insert documentation right into your code

```
def important_function(a=0.0, b=0.0):  
    """Do something important.
```

<http://docs.python.org/distutils/index.html>

# Documentation

- Good documentation is important if anyone else needs to use your code, and when the inevitable time comes that you forget what it does as well
- Docstrings make it easy to insert documentation right into your code

```
def important_function(a=0.0, b=0.0):  
    """Do something important.
```

Keyword arguments:

<http://docs.python.org/distutils/index.html>

# Documentation

- Good documentation is important if anyone else needs to use your code, and when the inevitable time comes that you forget what it does as well
- Docstrings make it easy to insert documentation right into your code

```
def important_function(a=0.0, b=0.0):
    """Do something important.

    Keyword arguments:
    a -- the first argument (default 0.0)
```

<http://docs.python.org/distutils/index.html>

# Documentation

- Good documentation is important if anyone else needs to use your code, and when the inevitable time comes that you forget what it does as well
- Docstrings make it easy to insert documentation right into your code

```
def important_function(a=0.0, b=0.0):
    """Do something important.

    Keyword arguments:
    a -- the first argument (default 0.0)
    b -- the second argument (default 0.0)
```

<http://docs.python.org/distutils/index.html>

# Documentation

- Good documentation is important if anyone else needs to use your code, and when the inevitable time comes that you forget what it does as well
- Docstrings make it easy to insert documentation right into your code

```
def important_function(a=0.0, b=0.0):
    """Do something important.

    Keyword arguments:
    a -- the first argument (default 0.0)
    b -- the second argument (default 0.0)

    """
```

<http://docs.python.org/distutils/index.html>

# Documentation

- Good documentation is important if anyone else needs to use your code, and when the inevitable time comes that you forget what it does as well
- Docstrings make it easy to insert documentation right into your code

```
def important_function(a=0.0, b=0.0):
    """Do something important.

    Keyword arguments:
    a -- the first argument (default 0.0)
    b -- the second argument (default 0.0)

    """
    [ ... here is where you actually do something important ... ]
```

<http://docs.python.org/distutils/index.html>

# Documentation

# Documentation

Can get on-the fly help through iPython...

# Documentation

Can get on-the fly help through iPython...

```
In [6]: datetime.datetime?
```

# Documentation

Can get on-the fly help through iPython...

```
In [6]: datetime.datetime?  
Type:          type
```

# Documentation

Can get on-the fly help through iPython...

```
In [6]: datetime.datetime?  
Type:           type  
Base Class:    <type 'type'>
```

# Documentation

Can get on-the fly help through iPython...

```
In [6]: datetime.datetime?  
Type:          type  
Base Class:    <type 'type'>  
String Form:   <type 'datetime.datetime'>
```

# Documentation

Can get on-the fly help through iPython...

```
In [6]: datetime.datetime?  
Type:          type  
Base Class:    <type 'type'>  
String Form:   <type 'datetime.datetime'>  
Namespace:     Interactive
```

# Documentation

Can get on-the fly help through iPython...

```
In [6]: datetime.datetime?  
Type:          type  
Base Class:    <type 'type'>  
String Form:   <type 'datetime.datetime'>  
Namespace:     Interactive  
File:          /Library/Frameworks/Python.framework/Versions/6.0.0/lib/python2.6/lib-  
dynload/datetime.so
```

# Documentation

Can get on-the fly help through iPython...

```
In [6]: datetime.datetime?  
Type:          type  
Base Class:    <type 'type'>  
String Form:   <type 'datetime.datetime'>  
Namespace:     Interactive  
File:          /Library/Frameworks/Python.framework/Versions/6.0.0/lib/python2.6/lib-  
dynload/datetime.so  
Docstring:
```

# Documentation

Can get on-the fly help through iPython...

```
In [6]: datetime.datetime?  
Type:          type  
Base Class:    <type 'type'>  
String Form:   <type 'datetime.datetime'>  
Namespace:     Interactive  
File:          /Library/Frameworks/Python.framework/Versions/6.0.0/lib/python2.6/lib-  
dynload/datetime.so  
Docstring:  
    datetime(year, month, day[, hour[, minute[, second[, microsecond[, tzinfo]]]]])
```

# Documentation

Can get on-the fly help through iPython...

```
In [6]: datetime.datetime?  
Type:          type  
Base Class:    <type 'type'>  
String Form:   <type 'datetime.datetime'>  
Namespace:     Interactive  
File:          /Library/Frameworks/Python.framework/Versions/6.0.0/lib/python2.6/lib-  
dynload/datetime.so  
Docstring:  
    datetime(year, month, day[, hour[, minute[, second[, microsecond[, tzinfo]]]]])
```

# Documentation

Can get on-the fly help through iPython...

```
In [6]: datetime.datetime?  
Type:          type  
Base Class:    <type 'type'>  
String Form:   <type 'datetime.datetime'>  
Namespace:     Interactive  
File:          /Library/Frameworks/Python.framework/Versions/6.0.0/lib/python2.6/lib-  
dynload/datetime.so  
Docstring:  
    datetime(year, month, day[, hour[, minute[, second[, microsecond[,tzinfo]]]]])
```

The year, month and day arguments are required. tzinfo may be None, or an

# Documentation

Can get on-the fly help through iPython...

```
In [6]: datetime.datetime?  
Type:          type  
Base Class:    <type 'type'>  
String Form:   <type 'datetime.datetime'>  
Namespace:     Interactive  
File:          /Library/Frameworks/Python.framework/Versions/6.0.0/lib/python2.6/lib-  
dynload/datetime.so  
Docstring:  
    datetime(year, month, day[, hour[, minute[, second[, microsecond[, tzinfo]]]]])
```

The year, month and day arguments are required. tzinfo may be None, or an instance of a tzinfo subclass. The remaining arguments may be ints or longs.

# Documentation

Can get on-the fly help through iPython...

```
In [6]: datetime.datetime?  
Type:          type  
Base Class:    <type 'type'>  
String Form:   <type 'datetime.datetime'>  
Namespace:     Interactive  
File:          /Library/Frameworks/Python.framework/Versions/6.0.0/lib/python2.6/lib-  
dynload/datetime.so  
Docstring:  
    datetime(year, month, day[, hour[, minute[, second[, microsecond[,tzinfo]]]]])
```

The year, month and day arguments are required. tzinfo may be None, or an instance of a tzinfo subclass. The remaining arguments may be ints or longs.  
if imag == 0.0 and real == 0.0: return complex\_zero

# Documentation

Can get on-the fly help through iPython...

```
In [6]: datetime.datetime?  
Type:          type  
Base Class:    <type 'type'>  
String Form:   <type 'datetime.datetime'>  
Namespace:     Interactive  
File:          /Library/Frameworks/Python.framework/Versions/6.0.0/lib/python2.6/lib-  
dynload/datetime.so  
Docstring:  
    datetime(year, month, day[, hour[, minute[, second[, microsecond[,tzinfo]]]]])
```

The year, month and day arguments are required. tzinfo may be None, or an instance of a tzinfo subclass. The remaining arguments may be ints or longs.

```
if imag == 0.0 and real == 0.0: return complex_zero
```

```
...
```

# PEP8, pep8, pylint

- There is an official Python style guide, called PEP8:  
<http://www.python.org/dev/peps/pep-0008/>
- Guido sez: ‘code is read much more often than it is written’
  - especially true for scientific code
  - Python enables you to write readable code, so do it! The world will seem a happier place.
  - **pylint**, **pep8**, and similar tools tell you whether code is self-consistent, if sections are duplicated, and where/how you break the style rules
  - get them with **pip**, run them like:  
  \$ **pep8 my\_code.py**

# pylint

# pylint

```
$ sudo pip install pylint
[ ... progress report ... ]

$ pylint [my_poorly_written_code.py]
[ ... ]
C: 29,0: Line too long (84/80)
C: 32,0: Line too long (102/80)
C: 48,0: Line too long (113/80)
C: 51,0: Line too long (91/80)
C: 23,8:approx_Pi: Operator not followed by a space
      d +=1
          ^^
```

# pylint

```
$ sudo pip install pylint
[ ... progress report ... ]

$ pylint [my_poorly_written_code.py]
[ ... ]
C: 29,0: Line too long (84/80)
C: 32,0: Line too long (102/80)
C: 48,0: Line too long (113/80)
C: 51,0: Line too long (91/80)
C: 23,8:approx_Pi: Operator not followed by a space
      d +=1
      ^^
```

you can modify the settings with a `.pylintrc` file:

# pylint

```
$ sudo pip install pylint
[ ... progress report ... ]

$ pylint [my_poorly_written_code.py]
[ ... ]
C: 29,0: Line too long (84/80)
C: 32,0: Line too long (102/80)
C: 48,0: Line too long (113/80)
C: 51,0: Line too long (91/80)
C: 23,8:approx_Pi: Operator not followed by a space
      d +=1
      ^^
```

you can modify the settings with a `.pylintrc` file:

```
[ generate a .pylintrc file from default settings ]
$ pylint --generate-rcfile > ~/.pylintrc

[ modify this .pylintrc file to adjust settings ]
```

# Testing - nosetests

- nosetests is a program that runs any pre-defined tests you have written, and reports back the results
- within your code, add test functions that act as self-checks
- naming convention: test\*
- lay out everything you want your code to do beforehand, as a self check
  - test-driven development

# Testing - nosetests

- write test functions that use the `assert` command - it raises an `AssertionError` if it does not evaluate to true

# Testing - nosetests

- write test functions that use the `assert` command - it raises an `AssertionError` if it does not evaluate to true

```
[ within my code ]  
def square_me(x):  
    return x**2  
  
def test_1():  
    # test the simple square_me function  
    assert square_me(2.) == 4.  
  
def test_2():  
    # testing without an absolute equality  
    assert abs(square_me(2.34) - 5.4756) < .001
```

# Outline

- Managing Packages -  
`pip, setup.py, virtualenv`
  - Command Line Parsing - `argparse`
  - Building Modules & Packages  
  
`<breakout session>`
  - Version Control - `git`
  - Debugging & Testing - `pdb, ipy %debug, pylint, pep8`
- Distribution - `distutils2`

- **Distributions** - `distutils2`

<http://pypi.python.org/pypi/Distutils2>

# • **Distributions** - *distutils2*

- *distutils2*: the standard way to take your directories of code and bundle them up for easy installation and use by others

<http://pypi.python.org/pypi/Distutils2>

# • Distributions - `distutils2`

- `distutils2`: the standard way to take your directories of code and bundle them up for easy installation and use by others
- You create a `setup.py` file which allows others to install your code in the standard fashion.

<http://pypi.python.org/pypi/Distutils2>

# • Distributions - `distutils2`

- `distutils2`: the standard way to take your directories of code and bundle them up for easy installation and use by others
- You create a `setup.py` file which allows others to install your code in the standard fashion.
- There are myriad options for the metadata you can define, an incredibly simple example is below:

<http://pypi.python.org/pypi/Distutils2>

# • Distributions - *distutils2*

- *distutils2*: the standard way to take your directories of code and bundle them up for easy installation and use by others
- You create a `setup.py` file which allows others to install your code in the standard fashion.
- There are myriad options for the metadata you can define, an incredibly simple example is below:

```
from distutils2.core import setup
```

<http://pypi.python.org/pypi/Distutils2>

# • Distributions - `distutils2`

- `distutils2`: the standard way to take your directories of code and bundle them up for easy installation and use by others
- You create a `setup.py` file which allows others to install your code in the standard fashion.
- There are myriad options for the metadata you can define, an incredibly simple example is below:

```
from distutils2.core import setup  
setup(name='My_Package',
```

<http://pypi.python.org/pypi/Distutils2>

# • Distributions - `distutils2`

- `distutils2`: the standard way to take your directories of code and bundle them up for easy installation and use by others
- You create a `setup.py` file which allows others to install your code in the standard fashion.
- There are myriad options for the metadata you can define, an incredibly simple example is below:

```
from distutils2.core import setup
setup(name='My_Package',
      version='0.01',
```

<http://pypi.python.org/pypi/Distutils2>

# • Distributions - `distutils2`

- `distutils2`: the standard way to take your directories of code and bundle them up for easy installation and use by others
- You create a `setup.py` file which allows others to install your code in the standard fashion.
- There are myriad options for the metadata you can define, an incredibly simple example is below:

```
from distutils2.core import setup
setup(name='My_Package',
      version='0.01',
      license='License_to_not_kill',
```

<http://pypi.python.org/pypi/Distutils2>

# • Distributions - distutils2

- **distutils2**: the standard way to take your directories of code and bundle them up for easy installation and use by others
- You create a **setup.py** file which allows others to install your code in the standard fashion.
- There are myriad options for the metadata you can define, an incredibly simple example is below:

```
from distutils2.core import setup
setup(name='My_Package',
      version='0.01',
      license='License_to_not_kill',
      py_modules=[ 'my_package_name' ],
```

<http://pypi.python.org/pypi/Distutils2>

# • Distributions - `distutils2`

- `distutils2`: the standard way to take your directories of code and bundle them up for easy installation and use by others
- You create a `setup.py` file which allows others to install your code in the standard fashion.
- There are myriad options for the metadata you can define, an incredibly simple example is below:

```
from distutils2.core import setup
setup(name='My_Package',
      version='0.01',
      license='License_to_not_kill',
      py_modules=[ 'my_package_name'],
      )
```

<http://pypi.python.org/pypi/Distutils2>

# •Distributions

Running the setup.py will install your package (and modules)  
into the python path

# •Distributions

Standard hierarchy includes several ancillary files, as well as the package and modules.

Running the setup.py will install your package (and modules) into the python path

# •Distributions

Standard hierarchy includes several ancillary files, as well as the package and modules.

A diagram showing a folder structure. At the top level is a dark gray rectangle labeled "My\_Package\_folder/". Inside it, at the same level, is a smaller white rectangle labeled "README.txt".

```
My_Package_folder/
 README.txt
```

Running the setup.py will install your package (and modules) into the python path

# •Distributions

Standard hierarchy includes several ancillary files, as well as the package and modules.

```
My_Package_folder/  
  README.txt  
  LICENSE.txt
```

Running the setup.py will install your package (and modules) into the python path

# •Distributions

Standard hierarchy includes several ancillary files, as well as the package and modules.

```
My_Package_folder/  
  README.txt  
  LICENSE.txt  
  setup.py  
  my_package_name/
```

Running the setup.py will install your package (and modules) into the python path

# •Distributions

Standard hierarchy includes several ancillary files, as well as the package and modules.

```
My_Package_folder/  
  README.txt  
  LICENSE.txt  
  setup.py  
  my_package_name/  
    __init__.py
```

Running the setup.py will install your package (and modules) into the python path

# •Distributions

Standard hierarchy includes several ancillary files, as well as the package and modules.

```
My_Package_folder/
 README.txt
 LICENSE.txt
 setup.py
 my_package_name/
   __init__.py
   my_module.py
```

Running the setup.py will install your package (and modules) into the python path

# •Distributions

Standard hierarchy includes several ancillary files, as well as the package and modules.

```
My_Package_folder/
 README.txt
 LICENSE.txt
 setup.py
 my_package_name/
     __init__.py
     my_module.py
     my_other_module.py
```

Running the setup.py will install your package (and modules) into the python path

# •Distributions

For a comprehensive guide to releasing code,  
check out:

<http://http://guide.python-distribute.org/>

# •Distributions

If you've written your own package, use distutils2 to create a standard, share-able zipped file

For a comprehensive guide to releasing code, check out:

<http://http://guide.python-distribute.org/>

# •Distributions

If you've written your own package, use distutils2 to create a standard, share-able zipped file

```
$ cd My_Package_folder
```

For a comprehensive guide to releasing code, check out:

<http://http://guide.python-distribute.org/>

# •Distributions

If you've written your own package, use distutils2 to create a standard, share-able zipped file

```
$ cd My_Package_folder  
$ python setup.py sdist
```

For a comprehensive guide to releasing code,  
check out:

<http://http://guide.python-distribute.org/>

# •Distributions

If you've written your own package, use distutils2 to create a standard, share-able zipped file

```
$ cd My_Package_folder  
$ python setup.py sdist  
  
[... created my_package_name-0.1.tar.gz ...]
```

For a comprehensive guide to releasing code,  
check out:

<http://http://guide.python-distribute.org/>

# Homework I

**Due Sunday, September 16th, 5:00pm, through an online git repository (see below)**

I) Get an account on GitHub or BitBucket (or some other equivalent service) and use Git to track your code, and push to this repository.

- get a 'private' repository if you'd like, to keep the internet from sniffing you out  
(free GitHub micro accounts are available for students with a .edu account, otherwise privacy costs \$\$.  
BitBucket, however, allows private repos for all free users. Both allow public repos for free.)
- if you have a private repo, find Isaac on GitHub or BitBucket, and share your repo with him (account: ishivvers)
- while writing your solution, make at least 5 separate commits with real comments (we will check history)
- push your final version before 5:00pm Sunday, and send Isaac an email explaining exactly how to clone it
- from here on out, log your homework in the same Git repo, and we will run a script that pulls the repository every Sunday at 5:00. This is how your future homeworks will be submitted!

# Homework I

2) Write a module called ‘CalCalc’, with a method called ‘calculate’ that evaluates any string passed to it, and can be used from either the command line (using argparse with reasonable flags) or imported within Python. Feel free to use something like eval(), but be aware of some of the nasty things it can do, and make sure it doesn’t have too much power: [http://nedbatchelder.com/blog/201206/eval\\_really\\_is\\_dangerous.html](http://nedbatchelder.com/blog/201206/eval_really_is_dangerous.html)

EXAMPLE:

```
$ python CalCalc.py -s '34*28'  
$ 952  
---- AND, from within Python ----  
>>> from CalCalc import calculate  
>>> calculate('34*20')  
>>> 952
```

2a) To make this more awesome, have your function interact with the Wolfram|Alpha API to ask it what it thinks of the difficult questions. NOTE: this added bit will only be worth 1 point out of 12, but it makes the program way way way more awesome.

To make this work, experiment with urllib2 and a URL like this:

'<http://api.wolframalpha.com/v2/query?input=XXXXXX&appid=UAGAWR-3X6Y8W777Q>'

where you replace the XXXXX with what you want to know. NOTE: the ‘&appid=UAGAWR-3X6Y8W777Q’ part is vital; it is a W|A AppID I got for the class. Feel free to use that one, or you can get your own and read more about the API, here: <http://products.wolframalpha.com/api/>

And you can explore how it works here: <http://products.wolframalpha.com/api/explorer.html>

EXAMPLE:

```
$ python CalCalc.py 'mass of the moon in kg'  
$ 7.3459e+22  
---- AND, from within Python ----  
>>> from CalCalc import calculate  
>>> calculate('mass of the moon in kg', return_float=True) * 10  
>>> 7.3459e+23
```

# Homework I

3) Include a setup.py and turn your module into a proper Python Distribution, so that we can install it and use it. (Test yourself, by creating a virtualenv and seeing if you can install and use it!)

Example Folder Hierarchy:

```
Your_Homework1_Folder/CalCalc.py  
Your_Homework1_Folder/setup.py  
Your_Homework1_Folder/README.txt
```

Include at least 5 test functions in CalCalc.py, and test with nosetests, to make sure it behaves the way you think it should.

EXAMPLE:

```
CalCalc.py:  
[...]  
def calculate(...):  
    [...]  
  
def test_1():  
    assert abs(4. - calculate('2**2')) < .001
```

When grading, I will create a virtual environment and attempt to install your module by running:  
python setup.py install

And then I will test it using nosetests, and I will attempt to run it from the command line, and by importing it in Python. Explain everything in your README.txt

You will be graded on: Git usage, nosetests, packaging your code, and your code itself

# Homework I

## SOME SUGGESTIONS:

1) use “if \_\_name\_\_ == '\_\_main\_\_': ”

<http://docs.python.org/tutorial/modules.html>

2) regular expressions, though annoying at first, can be very powerful when parsing text:

<http://docs.python.org/library/re.html>

3) Though git is confusing at first, the effort is worth it. Read more about git online, and learn how to manage your programming workflow:

<http://www-cs-students.stanford.edu/~blynn/gitmagic/>