Gimp 3.0: The great free image editor rolls out a major update

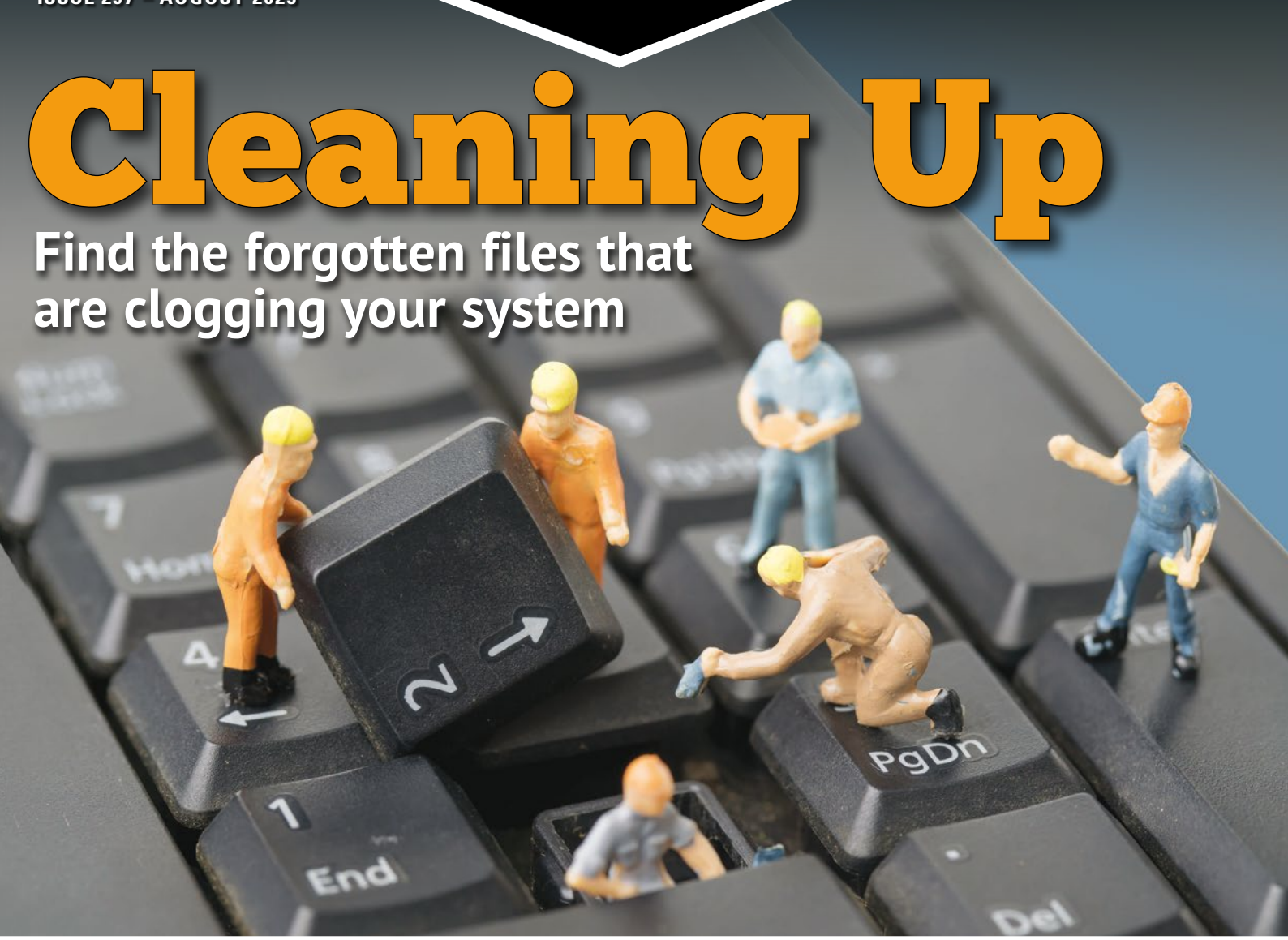
DVD INSIDE

LINUX MAGAZINE

ISSUE 297 – AUGUST 2025

Cleaning Up

Find the forgotten files that are clogging your system



ntfy: Push network alerts to your smartphone

Play music with a Pi Pico

notify-send: Get updates from remote Rasp Pis and Arduinos

ChimeraOS: Play Steam, GoG, and Epic games with this versatile gaming OS

Inform: Interactive fiction in Linux

10 TANTALIZING FREE APPS!

LINUX NEW MEDIA
The Pulse of Open Source

WWW.LINUX-MAGAZINE.COM

Create interactive fiction with Inform

Storyteller

Inform is your guide to the strange worlds of interactive fiction and text-driven games. BY JASON MCINTOSH

Interactive fiction (IF) is one of the most venerable super genres of digital creativity, an art form that has existed for almost as long as computers have been able to send and receive text. With its deepest roots in mainframe-and-teletype entertainments from the 1960s and 1970s such as Hunt the Wumpus and ELIZA, IF today encompasses hypertext games, visual novels, and a swirling variety of commercial and experimental work that focuses on text as its core interactive element.

Ages of Interactive Text

For much of the 1980s, IF often defined the cutting edge of computer games. A typical IF game of that era often adapted the most prominent UI of that era, one that's quite familiar to readers of this magazine: the command-line prompt. Titles such as Zork and Wishbringer published by Infocom, or The Hobbit by Melbourne House, let players explore entirely text-based worlds on their home computers, typing in verb-first text commands such as `GO NORTH`, `GET LAMP`, or `ASK MATILDA ABOUT GENETIC RECOMBINATION` for the game to parse and respond to. After the commercial viability of these "parser games" dried up in the 1990s, a hobbyist community formed on the young Internet to keep this archaic art form alive, developing tools and sharing resources that allowed contemporary audiences to continue playing these games – and making new ones.

In the decades since, the vibrant IF community has invented many playful technologies inspired by the parser games of yore, including the accessible hypertext kit Twine [1] and the narrative scripting language Ink [2]. These tools allow rapid development using modern design principles and game-engine integration, and some have gone on to commercial success. And yet the venerable parser game remains dear to the IF community, which often celebrates new games made in the old style, even if their audience is relatively limited. I like to think of a new parser IF game as a fresh jazz composition, or a new poem: Not everyone might appreciate it, but it has an unmatched power to uplift and inspire those who do.

(Or, thinking of an old saying about The Velvet Underground, while not many people discover parser IF, everyone who does launches their own game studio.)

The Role of Inform

One technology that has persevered from the earliest days of the IF community is Inform [3], a language and development environment for creating parser games. Originally released by Graham Nelson in 1993, it iterated through a complete rewrite with its version 7 in 2006 and, since 2022, has existed as an Artistic License 2.0 open source project whose development Nelson continues to lead.

Inform's major version number has since grown into double digits, but its user community still calls it "Inform 7" when they want to distinguish post-2006 Inform from its very different predecessors. Version 7 of Inform introduced two new technologies: a remarkable natural-language syntax that Nelson designed around Donald Knuth's "literate programming" paradigm and a book-shaped IDE with a focus on rich documentation and an almost eccentric level of chatty expressiveness in its terminal output and error reporting.

An Example Inform Game

Some masterpieces of parser-driven fiction, such as Emily Short's intrigue-laden Counterfeit Monkey or Michael S. Gentry's Lovecraftian epic Anchorhead, describe complex worlds of locations and characters and interwoven plots that can require many hours of focused play to navigate through.

And then there's the example game I wrote for this article, Penguin Security Protocols, that you can play through in about one minute. It has only one scene, one location, one actor, and a victory condition that most players would not find particularly hard to accomplish. Even so, it presents a glimpse of what a parser IF made with Inform

looks like – and, more pertinent for this article, what its source code looks like.

First I'll present a complete run through of the game in action. In the following transcript, the bold statements preceded with a › character represents player input. The rest is the game's textual output.

The Transcript

After a long morning of system administration, it's time for a well-earned nap. But even sleeping penguins need to mind basic operational security!

Server Room

A chilly room filled with the gentle hum of Linux servers. The air conditioning keeps things at a perfect penguin-friendly temperature.

Your Linux workstation sits against the wall, running smoothly.

The monitor displays your Linux desktop.

A silvery fish rests on a small plate.

A comfortable beanbag chair sits in the corner, perfect for penguin naps.

> listen to the servers

These powerful machines run various critical services. Their cooling fans create a constant white noise that's surprisingly soothing to a penguin's ears.

> sleep

That herring looks awfully tasty. Shouldn't you attend to it before your nap?

> look at the herring

A plump, fresh herring – a perfect penguin treat. Its scales glisten under the server room lights.

> eat herring

You're stopped short when you remember the first rule of penguin security protocols: Never leave your workstation unprotected while eating or sleeping!

> examine the workstation

A powerful machine running your favorite Linux distribution. The monitor displays a terminal window with several SSH sessions open.

The workstation is currently switched on.

> turn off workstation

You shouldn't turn off your workstation completely – you have important penguin work to do later!

Perhaps turning on the screensaver would be more appropriate.

> turn on screensaver

You quickly press the keyboard shortcut to activate your screensaver. Good security-conscious penguin!

> eat the herring

(first taking the herring)

The herring is delicious! Your tummy is now full and you feel sleepy.

> sleep

You settle into the beanbag with a satisfied penguin sigh. With a full belly and your workstation properly secured, you drift off to a well-deserved nap. Victory for security-conscious penguins everywhere!

The Source

The preceding transcript is my lead-in to letting you see the complete source code listing, which is in the box entitled "The Story."

If this somewhat fishy snack nonetheless whets your appetite for more, then let's talk about how to get Inform installed on your own Linux machine and then take a brief tour of its unusual IDE.

Install Inform on Linux

Because Inform is a mature open source project, you have several installation options of varying complexity.

If you have Flatpak set up on your system, then you can install Inform with a click or two through Flathub. For example, on a stock KDE Plasma setup, you can find a one-step Inform installer by clicking the *Discover* icon in your desktop panel then searching for *Inform*. Or, from a terminal window, run the following command:

```
flatpak install flathub com.inform7.IDE
```

You can also directly download `.flatpak`, `.deb`, or `.rpm` packages from the Inform releases page on GitHub [4].

Of course, you can always attempt to download, compile, and install Inform from source, but this will likely prove quite challenging, for several reasons. First of all, building Inform is a project in itself, requiring you to prepare a number of custom intermediate tools for your system before you can actually build the compiler. And beyond that, the IDE for every OS supported by Inform exists as its own software project, with its own maintenance team. Philip Chimento leads the Inform-on-Linux project, with its repository at GitHub [5].

If you want to try the very latest pre-release features and tweaks, or if you hunger with technical curiosity for how Inform is built, then by all means explore the two repositories' source code! The core Inform source, in particular, can make for a fascinating software-architectural study, because Inform itself is developed under the very same literate-programming philosophies that it applies to its language's syntax.

To actually get started creating with Inform, however, one of the pre-built packages probably suits your needs better.

Explore the Inform IDE

To see the Inform IDE from a fresh start, launch the Inform application and create a new Inform project by following the series of dialogs that it presents. This brings you to the full, twin-paned

IDE window, resembling Figure 1. By default, the left pane contains your project's source code, and the right pane shows the title page to Inform's deep documentation.

Read (and search) the docs

If you're already intrigued, then at this point you may want to pause and click around that documentation a little. As the title page makes clear, the documentation contains two separate but intertwined volumes:

- *Writing with Inform*, an explanatory guide to the whole system, including the language syntax, operating the IDE, and development workflows
- *The Inform Recipe Book*, a catalog of Inform code snippets demonstrating the many core features of the language, as well as various combinatorial techniques that bring together these ingredients in interesting ways

You can start at the first page of *Writing with Inform* and start reading, if you want. You'll soon discover that the two volumes cross-reference one another frequently, and the *Examples* and *General Index* tabs at the top of the Documentation pane give you more ways to explore the docs.

The Story

```
When play begins, say "After a long morning of system administration, it's time for a well-earned nap. But even sleeping penguins need to mind basic operational security!"
Chapter 1 - The server room, and its sysadmin
The Server Room is a room. "A chilly room filled with the gentle hum of Linux servers. The air conditioning keeps things at a perfect penguin-friendly temperature."
Tux is a person in the Server Room. The player is Tux. The description of Tux is "You are a rotund penguin with a sleek black back and crisp white front. Your natural tuxedo makes you both elegant and perfectly dressed for system administration duties." Understand "penguin" or "admin" or "sysadmin" or "system administrator" as Tux.
The servers are scenery in the Server Room. The description is "These powerful machines run various critical services. Their cooling fans create a constant white noise that's surprisingly soothing to a penguin's ears." Understand "server" or "racks" or "machines" or "fans" as the servers. Instead of listening, try examining the servers.
Chapter 2 - Stuff that's in the server room
The workstation is a device in the Server Room. It is fixed in place and switched on. "Your Linux workstation sits against the wall, running smoothly." The description is "A powerful machine running your favorite Linux distribution. The monitor displays a terminal window with several SSH sessions open." Understand "computer" or "machine" or "terminal" or "pc" or "desktop" or "linux box" as the workstation.
The screensaver is a device in the Server Room. It is fixed in place and switched off. "The monitor displays your Linux desktop[if the screensaver is switched on]. The screen is currently protected by your favorite animated penguin screensaver[end if]." The description is "Your screensaver features animated penguins sliding across ice floes. Not only is it adorable, but it also prevents unauthorized access to your system."
The herring is an edible thing in the Server Room. "A silvery fish rests on a small plate." The description is "A plump, fresh herring - a perfect penguin treat. Its scales glisten under the server room lights." Understand "fish" or "snack" or "food" or "treat" or "lunch" or "silver fish" as the herring.
The blue beanbag is a supporter in the Server Room. "A comfortable beanbag chair sits in the corner, perfect for penguin naps." It is enterable. The description is "A large, navy blue beanbag chair. It's positioned away from the server fans for optimal napping conditions." Understand "chair" or "bean bag" or "seat" or "bed" as the beanbag. Instead of sleeping, try entering the beanbag.
Chapter 3 - What you can do with all that stuff
Instead of switching off the workstation: say "You shouldn't turn off your workstation completely - you have important penguin work to do later! Perhaps turning on the screensaver would be more appropriate.";
After switching on the screensaver: say "You quickly press the keyboard shortcut to activate your screensaver. Good security-conscious penguin!".
After switching off the screensaver: say "You deactivate the screensaver with your secure passphrase.".
Instead of eating the herring when the screensaver is switched off, or entering the beanbag when the screensaver is switched off: say "You're stopped short when you remember the first rule of penguin security protocols: never leave your workstation unprotected while eating or sleeping!";
Instead of entering the beanbag when the herring is in the server room: say "That herring looks awfully tasty. Shouldn't you attend to it before your nap?"
After eating the herring when the screensaver is switched on: say "The herring is delicious! Your tummy is now full and you feel sleepy.";
After entering the beanbag when the screensaver is switched on and the herring is not carried and the herring is not in the Server Room: say "You settle into the beanbag with a satisfied penguin sigh. With a full belly and your workstation properly secured, you drift off to a well-deserved nap. Victory for security-conscious penguins everywhere!"; end the story finally saying "You've maintained proper security protocols!";
After entering the beanbag when the screensaver is switched on and the player carries the herring: say "You can't get comfortable with a fish in your flippers. Maybe you should eat it first?";
Test winning with "switch on screensaver / take herring / eat herring / get on beanbag".
```

Each item in *The Inform Recipe Book* is actually the complete ready-to-compile source to a tiny IF game, with controls that instantly copy the code into your source pane. This allows rapid hands-on experimentation and iteration while you learn the system.

A dedicated search bar at the top of the IDE window rounds out Inform's documentation UI, working much as you'd expect. However, the *General Index* tab is worth your time to get familiar with. It contains a remarkable and carefully curated traditional index of every significant concept raised in the documentation, arranged alphabetically – that is, in the fashion of a traditional index in the back of the print edition of a thorough reference manual. Beyond being a most unusual artifact to discover in purely digital documentation, this index gives a hint of the relaxed and scholarly attitude that Inform IDE expresses throughout the creative process.

Inform tries to keep you in the mindset of a writer working with human language, producing a work that you intend other humans to understand and enjoy, with a layer of machine interpretation in the middle. As much as possible, the IDE favors thoughtful expressiveness, and even a sort of rambling exploration, rather than a more typical programming environment's stance of maximizing digital efficiency.

Tour the other tabs

Let's look at the other tabs that run down the "spine" of the two IDE panes:

- *Results* displays the compiler's rather chatty terminal output from its most recent attempt to compile your story source.
- *Index* gives you insight into the way that Inform sees your story, after you've compiled it. You can flip through lists of rules, scenes, characters, and objects that your game has defined and view some cataloguing metadata that Inform automatically generates for your work.
- *Skein* and *Transcript* help you define, tune, and debug the paths that players should be able to take through your story, making sure that the game's reactions and responses to various text inputs match what you expect. Figure 2 shows my own development skein for Penguin Security Protocols. (A "real" game's skein is much more tangled than this simple sample.)
- *Extensions* lists community-authored extensions to the Inform language that you have installed. Inform for Linux ships with over a dozen extensions, and you can find more online. (The IDE does have a *Public Library* button intended to let you more easily browse available extensions, but at the time of this writing, it doesn't always work well with Linux IDE – particularly when installed through Flatpak.)

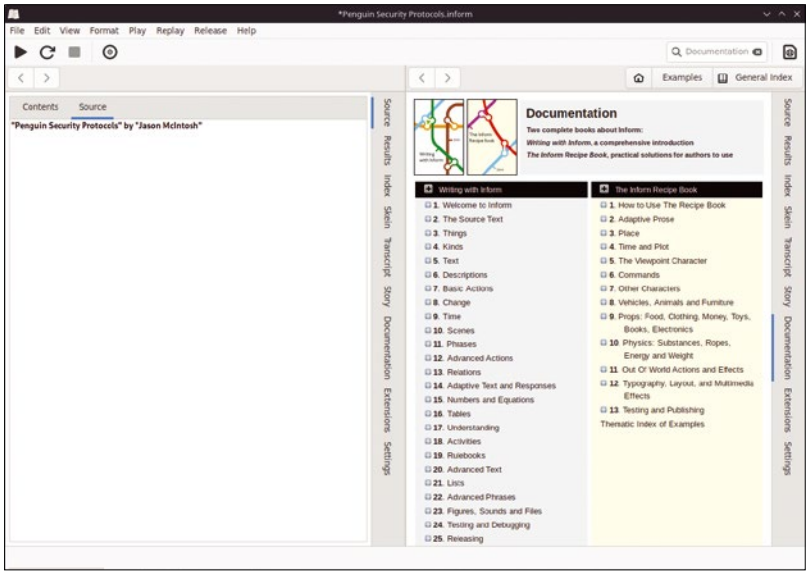


Figure 1: The Inform IDE with a brand new project loaded up.

- Finally, *Settings* lets you fine-tune various behaviors of the compiler, including an option to lock down the game's randomizer seed while running tests.

For a much more thorough guide to how all of these work, see the Inform documentation.

A typical Inform development workflow

Similar to any software development project, you spend much of your time with the default two panes open: your story source in one and the system documentation in the other.

As you start to build out your world, though, you'll spend more time with the Skein and Transcript panes visible, letting you carefully grow and trim a dangling spider plant of nodes and paths, one that represents all the major routes a player can traverse through your interactive story.

With the skein, you can leap to any node in a click, letting you fast-forward through time and space within your own story-world to help you

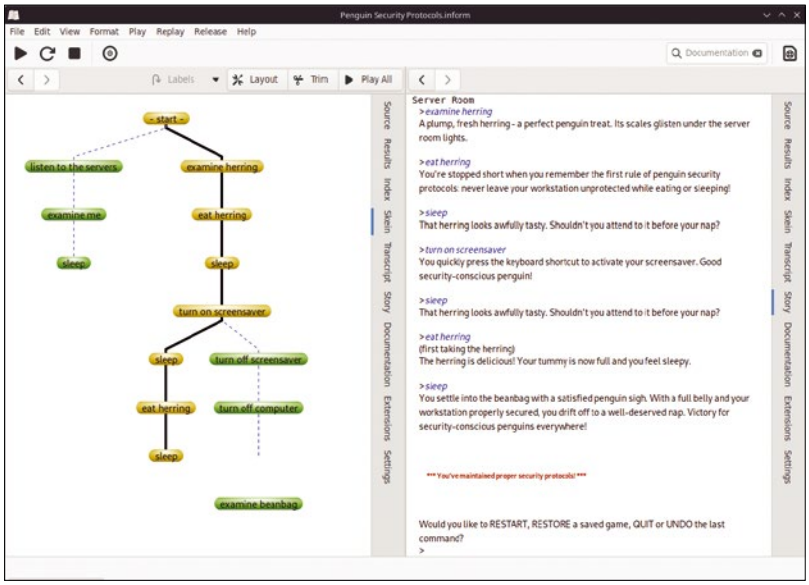


Figure 2: One possible skein of the sample game's branching paths.

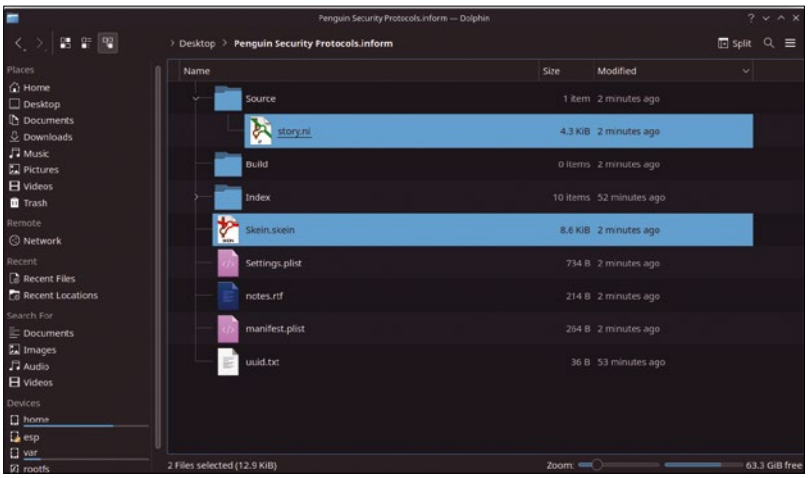


Figure 3: The minimum files to check into version control.

further test and develop it. By pairing the skein with the *Transcript* tab, you can ensure that each scrap of conditionally displayed text that you've written gets displayed exactly when and where you expect, and under the correct circumstances.

Invariably, this means lots of playtesting. Like any kind of game development, the interactive fiction workflow ideally has you letting trusted testers play with early drafts of your work, finding flaws and suggesting improvements. In particular, parser-based IF is infamous for easily falling into a "guess the verb" trap, where players try inputs that seem quite reasonable within the fiction, but which the author hadn't foreseen, leading to a disappointingly generic and frustrating "You can't do that" response. A polished parser game anticipates a wide variety of reasonable inputs, helping the player feel understood and engaged.

In my own experience as an IF author, the skein of a game's final draft largely comprises pathways I added based on testers' suggestions, letting the game respond satisfyingly to curious and thoughtful players as they poke and prod around the little world of words that I've built.

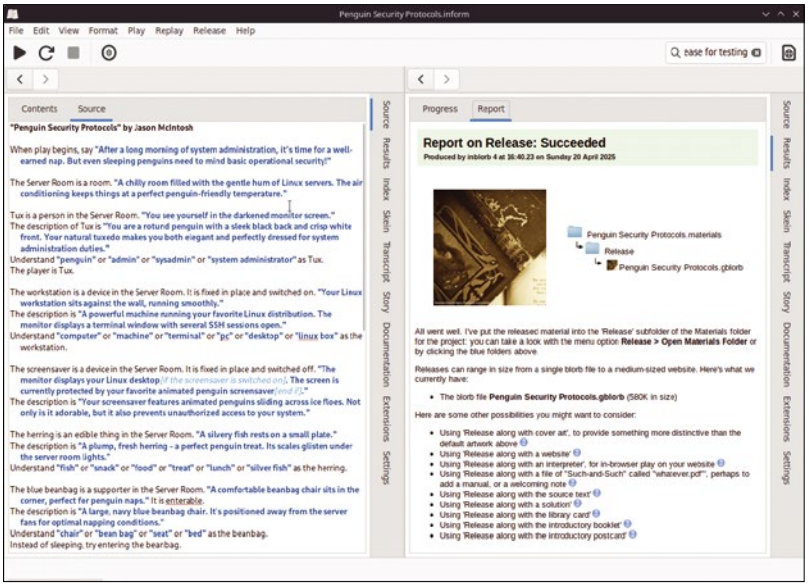


Figure 4: Releasing Penguin Security Protocols.

Inform and version control

While the IDE lacks any built-in version control support, you can use your usual version-control workflow with an Inform project.

As illustrated by Figure 3, your Inform project has two files worth checking into your version control system, both located in the `.inform` directory that you created the first time you saved your project:

- `Source/story.ni` is a text file containing the source code of your story
- `Skein.skein` is an XML file defining your game's skein and transcript data, crucial for designing your testable pathways through the work

Other files that you might consider adding to version control include the following, especially if you plan to share your source or otherwise build the project on more than one machine:

- Any multimedia files, such as images or sound effects, that you add to your game's resources
- `uuid.txt`, containing a UUID that Inform generates for your game's catalog metadata
- Any Inform extensions that you import into your game, found in `PROJECT_NAME.materials/Extensions`, to enforce consistent versioning for those extensions

The other files and directories that make up your project are generated by Inform as you compile or test your game and don't benefit much from additional version control.

Release and Distribute Your Game

When you're ready to start letting others explore your interactive story, Inform gives you plenty of options for creating playable release files. And when you're ready to find a larger audience – as well as a community of willing testers and friendly fellow authors – you can take advantage of many free resources for perfecting your work and then sharing it with the world.

Create a release

Inform makes sharing an independently playable copy of your work relatively simple: Press the *Release* button at the top of the IDE. If the game successfully compiles, then Inform creates release files and tells you all about it, as shown in Figure 4.

By default, the format of an Inform release is a single file in `.blorb` format, which you (or your testers) can play using a dedicated IF interpreter application, such as the modern and graphical Lectrote [6] or the terminal-friendly throwback Frotz [7].

However, as Figure 4 also illustrates, Inform invites you to add various imperative sentences to your source in order to trigger additional release formats and options. In particular, telling Inform

to `Release along with a website and an interpreter` prompts it to publish your game as a fully self-contained website, ready to publish on the web or even just send to someone for local play. Testers and players often find this most convenient, because it lets them play your game in any modern browser without the need to find and install a separate interpreter application.

Find playtesters, audience, and inspiration

The Interactive Fiction Community Forum [8] is a lively hub for discussing and sharing all things IF. Operated and moderated by volunteers from the nonprofit Interactive Fiction Technology Foundation, the forum offers a number of spaces for discussing every aspect of playing, sharing, and creating IF work. This includes sub-forums for learning specific creative technologies – Inform is most certainly included – and for finding playtesters.

From the forums, you can continue your dive into the rich world of IF resources by browsing the Interactive Fiction Database [9], a vast catalog of creative work that spans five decades and thousands of entries for play and study. This includes lots of titles written in Inform – including those with published source code that you can browse for further illumination with your own work.

To browse the sublimely readable source code of an Inform masterwork whose complexity sits at the polar extreme from this article's trivial example, see the public repository for Counterfeit Monkey [10]. This repository also serves as a model for files worth checking into version control, as discussed earlier in this article.

Your Journey Begins

Whether you're quite familiar with the Zork-ish text games of old, or whether you've never encountered language-driven video games quite like this before, I hope this article has inspired you to investigate the creative potential of Inform on Linux.

In a way, IF has a particular affinity with Linux: two technologies rooted deep in the previous century that continue to exert influence over their respective fields today. To be sure, the obscure underground empire of creativity and community offered by contemporary IF welcomes the sort of literate coders (or code-minded writers) that Linux tends to attract, as well! If you find yourself inhabiting this kind of creative crossover space, then I invite you to pick up a tool such as Inform and see what strange and wordy magic you can make, too. ■■■

Info

[1] Twine: *https://twinery.org*

[2] Ink: *https://www.inklestudios.com/ink/*

[3] Inform: *https://inform7.com*

[4] Inform releases: *https://github.com/ganelson/inform/releases*

[5] Inform-on-Linux project: *https://github.com/ptomato/inform7-ide*

[6] Lectrote: *https://github.com/erkyrath/lectrote*

[7] Frotz: *https://davidgriffith.gitlab.io/frotz/*

[8] Interactive Fiction Community Forum: *https://intfiction.org*

[9] Interactive Fiction Database: *https://ifdb.org*

[10] Counterfeit Monkey: *https://github.com/i7/counterfeit-monkey*

The Author

Jason McIntosh is a writer who lives in New York City. His personal website is *https://jmac.org*.

■■■